Adaptive Neural Networks Layer for Multi-Speaker Separation Problem

Multi-Speaker 問題に対する Adaptive Networks Layer についての研究

by

Anthony D'Amato
アントニー ダマト

A Master Thesis
修士論文

Submitted to
the Graduate School of the University of Tokyo
on July 11, 2018
in Partial Fulfillment of the Requirements
for the Degree of Master of Information Science and Technology
in Computer Science

Thesis Supervisor: Hiroshi Imai　今井 浩
Professor of Computer Science

ABSTRACT

The complexity of sound, its noisy character and the overlapped information makes the single channel Multi-Speaker Separation quite a challenge for the audio signal processing area. In comparison, humans are very good at distinguishing different voices in complex environments.

One of the first successful approach for this problem was to use non-negative matrix factorization (NMF), sparse NMF and later convolutional NMF but these methods had some drawbacks because of their linearity, cost and weak representation.

With the arrival of Deep Learning in audio signal processing, improvements have been made in denoising, audio classification, and mostly in Natural Language Processing (NLP) with the recent WaveNet and Tacotron 2 architectures. In the recent years new techniques involving Deep Learning were proposed and are performing quite well in the Multi-Speaker Separation problem. These latter are based on producing masks for mixed spectrograms to separate voices. To do so, these methods apply bidirectional recurrent neural networks on spectrograms trained to produce embeddings representing each speaker. This created embedded space allows to apply unsupervised algorithms such as k-means to segment the spectrograms and create binary masks. Contrary to the previous class-based segmentation methods, these new approaches are partition-based segmentation and lead to a speaker-independent inference and therefore easier generalization and precision. These approaches could reach more than 11dB improvement for the Signal to Distortion Ratio (SDR) in average for separating male and female mixed voices.

But these methods still struggle to separate mixed voices of the same gender and thus struggle to separate more than three voices from a single input channel. We presume that this problem might be mainly due to the overlapping frequencies in the mixed spectrograms when the same genders are speaking. To overcome this issue, we replace the use of spectrograms and propose a new approach consisting in using a sparse linear autoencoder, named 'adaptive layer', added to the current state of the art methods, namely the Deep Clustering (DPCL) and Source Contrastive Estimation (SCE) approaches. This autoencoder is composed of two main parts, a front-end layer and a back-end layer, and is able to take in input raw mixed audio signals and output separated audio signals.

In this study, we compare the SDR improvements with and without the adaptive layer for a mixture of different speakers. We show promising results with this new layer that can enhance Deep Clustering results. Furthermore, we improve the SCE approach using methods introduced by Deep Clustering and other new regularization methods. We improvement the SCE method by dB

Acknowledgements

Firstly, I would like to express my gratitude to my supervisor Hiroshi Imai for his support and wise advice all along this thesis, in addition, I would like to deeply thank Hidefumi Hiraishi and Naoto Ohsaka who helped me a lot over these years. I would also like to thank Doctor Kazuki Yoshizoe of RIKEN AIP for all his feedback and support, and for accepting me in his research unit as a trainee and allowing me to use RAIDEN computer without which I would not have been able to realize all this work.

Secondly, I would like to especially thank Natalia Jarzebska and Dea Luma for all their support throughout these 2 years and a half in Japan, having them by my side was a huge source of motivation and inspiration. Thank you for all your advice, for having listened to me, and for all the amazing moments we spent together, I will never forget everything you did for me.

Finally, I would like to express my profound gratitude to all my familly for all their support all over these years, even with the distance they continued to always be by my side and encouraging me to do my best.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Multi-Speaker Separation Problem

In 1953, Colin Cheery called the capacity of human beings to separate many voices in a conversation the *cocktail party problem*. We know that humans are very good at separating voices or other sounds from a noisy environment by focusing their attention to a particular chosen target. In this study, we focus our attention on the multi speaker separation problem consisting in separating from a **single channel** a mixture of speakers with different genders.

More generally, the source separation problem can be very useful in domains like automatic speech recognition, speech enhancement for hearing aids or in music tracks separation.

One challenge in the source separation problem is called the '*permutation problem',* that is happening when segmentation is used to separate the sources without having knowledge of the separated sources, therefore we do not know which partition belong to which source. This is a problem when chunking the input in order to process it in a separating model, the first chunk will output a certain order for the separation but it is not assured that next chunks will be outputted in the same order. We will discuss later in this study about how new deep learning architectures are solving this problem.

In this study, we only focus on the **single channel separation problem** but multi-channel audio separation is as well a very active area that has shown to produce even better results than using a unique channel. Today's deep learning methods, such as Deep Clustering [19, 24] can achieve quite good results in the multi speaker separation problem for 2 different genders speakers mixtures in terms of SDR improvements but are still not obtaining as successful results for **speakers of the same gender** or for a mixture of more than 2 speakers.

The main advantage brought by Deep Clustering [19] method is that it is a partition-based segmentation where the labels of the partitions are learned in opposition of a class-based segmentation where the labels of the segmented sources are learned. This approach allows to generalize very well to unknown speakers because it does not learn '*who*' is speaking but how to differentiate each speaker without the '*who*' information. Deep Clustering method produces discriminative embeddings from spectrogram magnitudes to apply clustering algorithm on them and then assign labels to each partitions. These learned partitions are used to create masks for the input spectrograms and separate each voice.

In this study we intend to improve the performances of current state of the art methods regarding same gender mixtures.

Firstly, this study is in part based on [47] which shows that using an end-to-end autoencoder architecture on raw audio mixture for the source separation

problem instead of Discrete Cosine Transform spectrograms can lead to better SDR improvement but this study is limited in terms of number of separated speakers, deep neural network complexity and generalization error. We suppose that a similar architecture, compatible with current state of the art deep learning methods and replacing the use of spectrograms, can lead to better results. In addition, we suppose that one problem with the use of spectrograms is that for a mixture of speakers with the same gender it is more likely that frequencies overlapping occurs, and thus separating information becomes more challenging. The architecture we propose to replace the use of spectrogramw tries to decrease this overlapping problem and thus intends to increase performances with mixtures of the same gender.

Secondly, we consider another approach named Source Contrastive Estimation (SCE) [45] that is based on the same principles than Deep Clustering and has shown to have better results. Indeed, since SCE method delivers better SDR improvement than the first version of Deep Clustering [19] and since the methods used in its second version [24] can be applied to any other methods, we presume that applying such improvements to SCE can lead to an enhancement of its performances.

1.2 Contributions

In our work we are adressing the improvement of the multi speaker separation problem with two main approaches:

Firstly, we propose a sparse autoencoder architecture to replace the use of spectrogram and lower the overlapping in the usual representation. We show that this neural network can improve the performances for 2 speakers mixtures but creating such an architecture for a mixture of 3 speakers is more challenging.

Secondly and in parallel, we improve the Source Contrastive Estimation method using different techniques brought by the second version of Deep Clustering [24]. In addition, we try new regularization techniques for SCE such as Negative Sampling or combining it with the DANet [7] method. We show that all these combinations can lead to significant improvements of the SCE method.

Chapter 2

Background

2.1 The Short Time Fourier Transform (STFT)

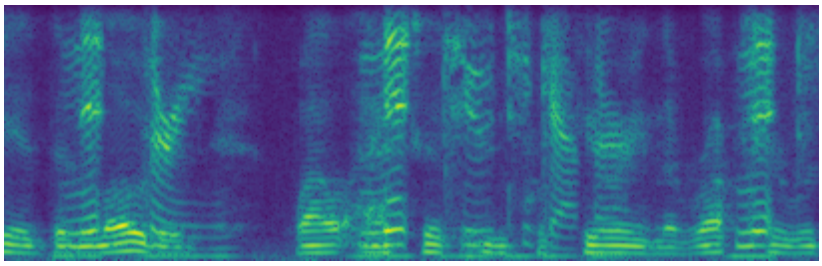


Figure 2.1: Spectrogram of a 2 speakers mixture using the square root of the Hann function as window of size 256 and a hop size of 64. The length of the audio signal used is of 26368 frames ($\simeq$ 3.3 seconds),and in this case $F = 129$ and $T = 409$.

Short-Time Fourier Transform (STFT) has played a significant role in signal processing research and has been broadly used in many applications involving sounds analysis. This transformation on a signal is obtained by applying the Fourier Transform along it on a fixed-sized window and moving it along with some overlapping. This operation converts this signal that has frequency changes over time into a time-frequency domain that can be more understable for analysis than its original 1-dimensional representation. The STFT of a signal x at the frame t and frequency f, named the *TF-bin*, is defined as follow:

$$STFT\{x\}(t,f) = X(t,f) = \sum_{n=-\infty}^{n=+\infty} x[n]w[t-mh]e^{-jf\frac{2\pi}{N}t} \in \mathbb{C}^{T\times F}$$

Here, f designates the frequency index, t is the frame center and h the hop size. The *analysis window* used for the tranformation is w and is of size N. The most commonly used analysis windows are the Hann and Hamming windows. The STFT of a signal x can be seen as a complex matrix $X \in \mathbb{C}^{T\times F}$ (T representing the time axis and and F the frequency axis) that can be decomposed as follow:

$$STFT\{x\} = X = |X| \exp^{j\phi}$$

In this decomposition, $|X|$ represents the STFT magnitude and ϕ its phase. In audio analysis, what is called the ***spectrogram*** (Figure 2.1) of an audio signal is the squared magnitude of the computed Short-Time Fourier Transform and is defined as follow:

$$spectrogram\{x\}(t,f) = |X(t,f)|^2$$

In this study, we consider a mixture of M speakers ,with the same or of a different gender, x defined as:

$$x = \sum_{i=0}^{M} x_i$$

Here, x is named the ***mixture*** and each x_i are the ***original sources***. Since the STFT is a linear operation, applying it on the mixture leads to:

$$STFT\{x\} = X = \sum_{i=0}^{M} X_i = \sum_{i=0}^{M} STFT\{x_i\}$$

Since complex numbers are difficult to handle in audio processing, the phase is most of the time discarded and only the magnitude is used. Even if some information might be lost by discarding the phase, studies [3] have shown that for small window size (20 - 40ms) most of the information is contained in the STFT magnitude. It has been as well shown that, even if the absolute value is taken into account, the additivity approximately holds regarding and thus that: $|X| \simeq \sum_{i=0}^{M} |X_i|$. This property and the non-negativity of spectrograms lead to the use of masks in order to perform source separation.

2.2 Spectrogram Masking

Masking is a technique broadly used for speech denoising or source separation. It consists in applying element-wisely a matrix $m^{T\times F\times M}$ to separate M sources from the mixture or $m \in R^{T\times F}$ for the speech enhancement case since it is only desired to separate noise from one specific source. Since the STFT representation for audio signals is sparse, it is likely that a specific source is dominant at a particular bin in terms of energy. This property and the non-negativity of the STFT magnitude motivated the use of masks in order to separate sources from an unique spectrogram.

The ***ideal binary mask*** $IBM(t,f)_i$ applied at a (t,f) bin for the i^{th} speaker is defined as:

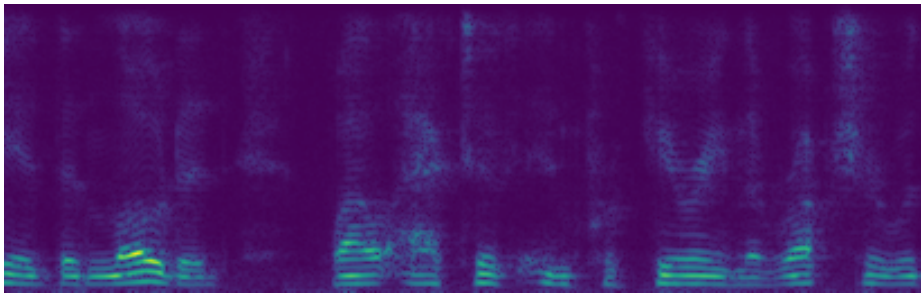
$$m_i^{ibm}(t,f) = IBM(t,f)_i = \begin{cases} 1 & \text{if } |X(t,f)_i| > max_{j\neq i}|X(t,f)_j \\ 0 & \text{else} \end{cases}$$

The ideal binary mask can be seen as a matrix $m^{ibm} \in R^{T\times F\times M}$ where an element at (t,f,i) is set to 1 for the i^{th} speaker if this latter is dominant compared to the others and 0 otherwise. In [42], it is shown that IBM can be very efficient for separating sources but highly depends on the STFT window size, and this optimal can vary according to the source type and task.

Another popular mask is the ***Wiener-like filter***, which generally leads to better Signal to Distortion Ratio (SDR) since it is a softer masks of which values lands in [0,1]. The Wiener-like filter is defined as:

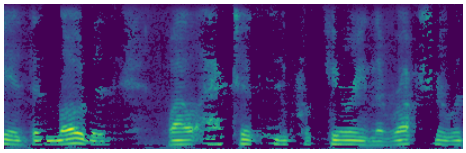
$$m_i^{wf} = \frac{|X_i|^2}{\sum_j |X_j|^2}$$

We show the difference between binary and Wiener-like masks in Figure 2.2. In the result section 6 we compare our results with the Wiener-like filters.

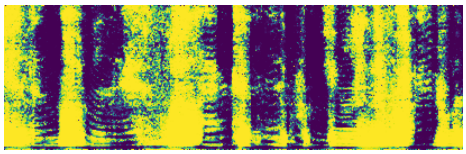


(a) Spectrogram of the 1^{st} speaker from 2.1

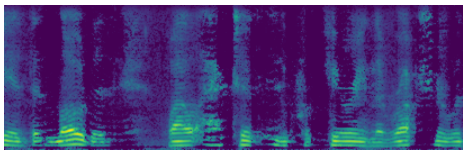
(b) Binary mask extracting the 1^{st} speaker



(c) Reconstruction using the binary mask



(d) Wiener-like mask extracting the 1^{st} speaker



(e) Reconstruction using the Wiener-like mask

Figure 2.2: Comparison between binary masks and Wiener-like masks. As mentioned, we can see that using Wiener-like (Figure 2.2d) that are softer than binary masks (Figure 2.2b) is leading to a better reconstruction

2.3 Deep Learning

Deep Learning [28, 13] has seen an explosion of interest these last years and is now applied in many fields in machine learning such as classification, clustering, generation or prediction. In this section we present basic deep learning concepts such as Feed Forward Networks (FFN), Convolutional Neural Networks (CNN) and Recurrent Neural Networks. Furthermore, we explain more complex structures that we encounter in this study like Bidirectional Long Short Term Memory (BLSTM) cells and sparse autoencoder.

2.3.1 Feed Forward Networks

Artificial Neural Networks (ANN) are based on the biological observation of human brain and how actual neurons networks are working. A human brain contains approximately 100 billions neurons that are connected to each other, studies say that an neuron has in average around 50000 connections with other neurons. They are connected to each other by what is called the *synapses*. Articial neurons are mimicking the process of biological neurons in two main steps: firstly, they take in input many other artificial neurons outputs and magnify it with a certain *weight*, secondly, depending on the outcome of this operation an activation function will determine how much of this output must be outputted to other neurons it is connected to. This process is explained in Figure 2.3 and can be formulated as:

$$h_i(x) = f(\sum_j w_{i,j} \cdot x_j + b_i)$$

$$h(x) = f(W \cdot x + b)$$

Here, W is named the weights of the neuron layer and b the biases. The function f is named the activation function and decides how much information is transmitted to the next layer. This operation can be repeated several times and thus form a multi-layered feed forward network, as well named fully connected layers since all the neurons is each layer are connected to each other. [22] has shown that this structure can approximate any functions. The intermediate layers are named the *hidden* layers and the last one is logically named the *output* layer.

Previously, the most commonly used activation functions were the sigmoid and the hyperbolic tangent, but because of their particular derivatives they caused either vanishing or exploding gradient. We explain this phenomenon in subsection 2.3.5.

$$sigmoid(x) = \frac{1}{1 + e^{-x}}$$

$$tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

The new commonly used activation functions are the Rectified Linear Unit (ReLU), it's leaky version (leaky ReLU) and the Exponential Linear unit. All these activation functions reduce the vanishing gradient problem encountered with the previous one, and in particular ReLU functions are used for their computation efficiency.

$$ReLU(x) = max(0, x)$$

$$ReLU_\alpha(x) = max(\alpha x, x), \alpha \leq 1$$

$$ELU(x) = \begin{cases} x & \text{if } x \geq 0 \\ \alpha(e^x - 1) & \text{otherwise} \end{cases}$$

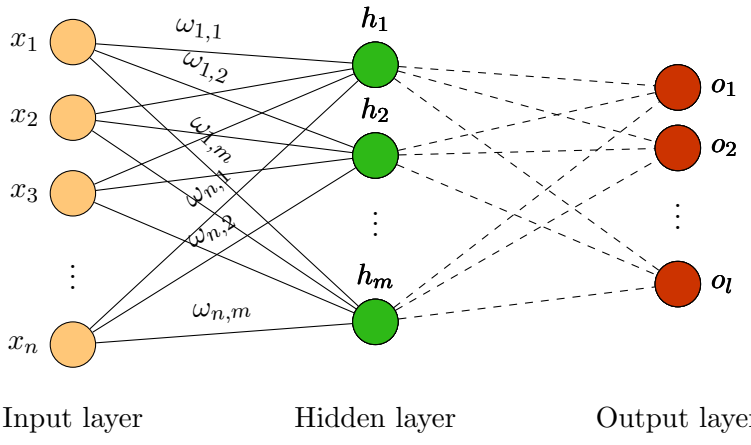


Figure 2.3: Example of a fully connected neural network with several hidden layers

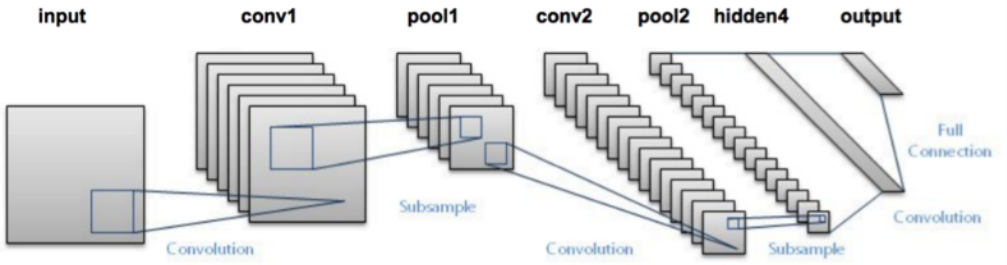


Figure 2.4: LeNet-5 convolutional network architecture introduced by Y. LeCun . In this figure, we see that CNN are made of an alternation between convolutional operations (with a non linear activation function applied) and subsampling operations. This architecture was used on digits recognition

2.3.2 Convolutional Neural Networks

Convolutional Neural Networks (CNN or ConvNet) [59, 17] (Figure 2.4) are networks made from the same artificial neurons presented previously, but do not apply the same operations as fully connected networks. The interest for CNNs increased a lot after AlexNet deep learning architecture won the ILSVRC competition in 2012 using this convolutional networks and improved previous state of the art method by almost 11%. From then, deeper and more complex CNN networks have been built and constantly improving their accuracy. The main difference with fully connected networks is that, as its name suggests, CNN are using convolutional operations with ***filters/kernels*** along their inputs in order to create a ***feature map/receptive field***.

Image analysis began to be very difficult to use with fully connected networks because, for 256×256 images for instance, the input has to be flatten and as a result the number of weights in the networks becomes very large. In CNN this problem is solved using filters that are shared across the receptive field, and thus considerably decreasing the number of weights used in the architecture.

Today's common CNN architectures, used for image analysis for instance, are made of blocks computing the following operations: first, a **convolution** operation is applied on the input with several filters on each channel of the input (for instance images can be made of red, green and blue channels), then a non-linear activation function is applied and finally a subsampling operation such as maxpooling or average pooling is applied, like it is shown in Figure 2.4.

In our study, with the Adaptive layer (section 4.1) we use one-dimensional convolution layers along audio signal in order to learn useful filter-banks for the multi speaker separation problem.

2.3.3 Recurrent Neural Networks

Compared to vanilla feed forward networks, recurrent neural networks (RNN) are ANNs with recurrent connections, they are made of hidden states evolving in a non-linear dynamic way and are mostly used with sequential learning.

$$h^t = f(W_h \cdot h^{t-1} + W_{xh} \cdot x^t + b_h)$$
$$y^t = g(W_{hy} \cdot h^t + b_{hy})$$

But one of drawback of Recurrent Neural Networks is that they have problem memorizing information or context for long sequences and this is mainly due to

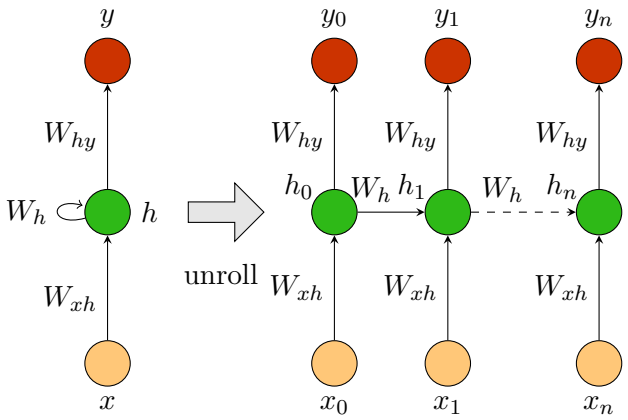


Figure 2.5: Example of a vanilla Recurrent Neural Network (RNN)

the vanishing gradient problem [37].

Long Short Term Memory architectures [21] (Figure 2.6) are solving this problem by using a "memory cell" that is transiting the main information using linear operations. As its name suggests, this architecture can learn long-term dependencies. The chained structure is kept but cells do not interact with the input the same way classical RNNs do.

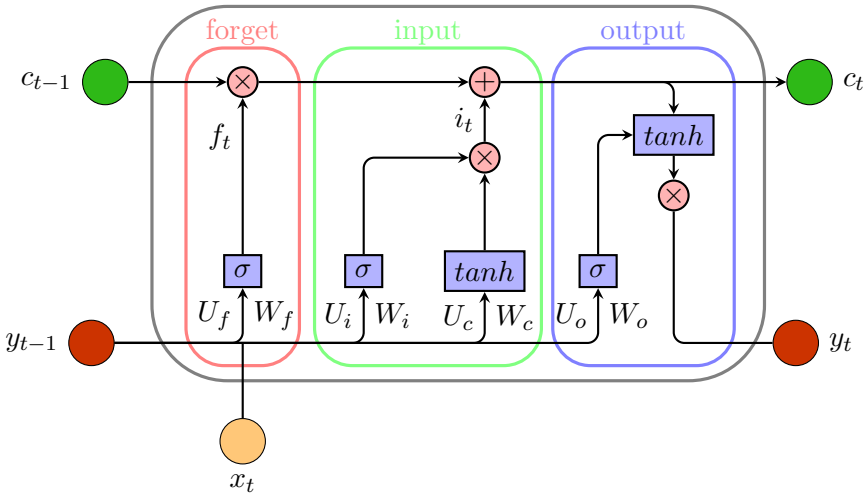


Figure 2.6: Architecture of a Long Short Term Memory cell (LSTM) - in red is the **forget gate**, in green the **input gate** and in blue the **output gate** - matrices on the left of lines are applied to y_{t-1} and the one on the right to x_t

LSTMs block consists in a **memory cell**, an **input gate**, a **forget gate** and an **output gate**. The **memory state** is a high dimensional state holding the information about past inputs and is updated according to its previous state and new inputs. The **forget gate** will decide what information in the current memory state the block should keep or delete. This forgetting operating is done as follow:

$$f_t = \sigma(W_f x_t + U_f h_{t-1} + b_f)$$

This forgetting vector is then applied in an element-wise way to the memory cell in order to keep or delete information.

The **input gate** is deciding what information to add to the memory cell

using two operations that are then multiplied together. The first one is applying a sigmoid function to the input in order to select from which part of the input the information will be added and the second is applying a hyperbolic tangente to determine the value added. Then this gate is added to the current memory state after the forget state has been applied:

$$i_t = \sigma(W_i x_t + U_i y_{t-1} + b_i) \circ \tanh(W_c x_t + U_c y_{t-1} + b_c)$$

The **output gate**, as its name implies, is finally deciding what is the output using the memory state and the current input. The input will choose what to output using a sigmoid and the memory state will decide the value by applying an hyperbolic tangente:

$$y_t = \sigma(W_o x_t + U_o y_{t-1} + b_o) \circ \tanh(c_t)$$

Therefore the main equation for the memory state is the following:

$$c_t = f_t \circ c_{t-1} + i_t \circ \tanh(W_c x_t + U_c y_{t-1} + b_c)$$

Bidirectional Long Short Term Memory (BLSTM) (Figure 2.7) is a combination of two regular LSTMs reading the same input sequence in two opposite directions. The output of these two LSTM networks are then concatenated to form a BLSTM. Bidirectional Long Short Term Memory are very efficient when it comes to handle input sequences that are not only interpretable linearly from their first element to their last but also have a global complex context.

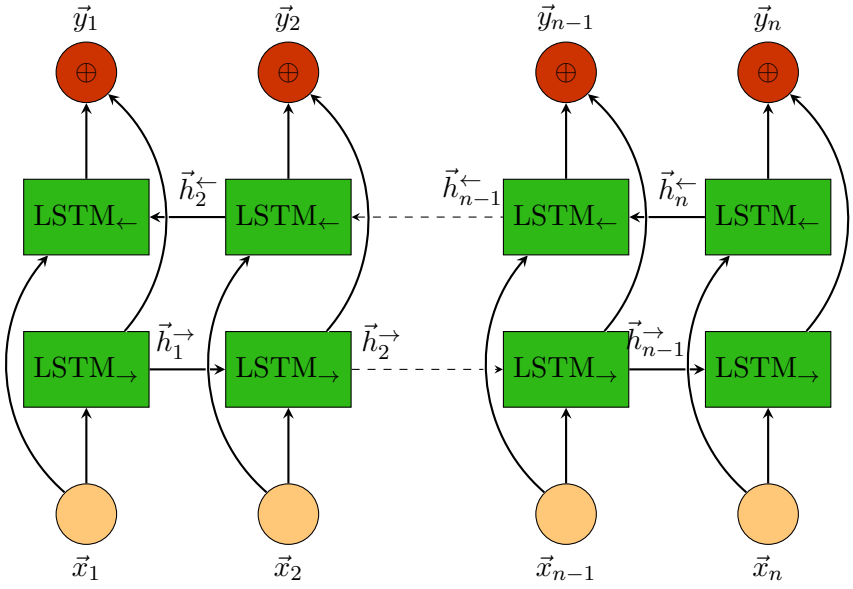


Figure 2.7: Bidirectional Long Short Term Memory network consists in two LSTM networks of opposite direction from which the outputs have been concatenated together.

We will see in our study that stacked BLSTM layers are very efficient for handling structures such as spectrograms and extract useful information to operate source separation.

2.3.4 Autoencoders and Sparsity

Autoencoders [5], also called auto-associators, are two layers perceptrons trained to reconstruct their input, and are in that sense called **self-supervised**. This particular architecture is trying to learn clever *representation* of the input by reducing or increasing (overcomplete) the number of hidden units compared to this latter. This is done to avoid learning the identity function (what would happen if the hidden layer has the same size) and can force the network to learn specific features of the input in order to reconstruct it efficiently. Typically, with an input x, a two-layered autoencoder (Figure 2.8) is defined as:

$$c(x) = f(W_1 \cdot x + b_1)$$

$$y = f(W_2 \cdot c + b_2)$$

With W and b the weight and biases of the network, and f the activation function. Here, c is named the **latent representation**, the **code layer** or the **bottleneck** of the autoencoder, with $dim(c) < dim(x)$ or $dim(c) > dim(x)$, this hidden layer is the one learning interesting representations of the inputs. Then, once the autoencoder is trained to reconstruct its input, unsupervised algorithms such as k-means can be applied to cluster the dataset used to train the network. Here we presented only a two-layered autoencoder but this architecture can certainly be extended to multiple layers and even use more complex operations such as convolutional or recurrent networks. Autoencoders are often divided in two main parts, the first one reducing the dimensionality of the input and the other one reconstructing it and increasing its dimensionality. They are often called the *encoder* and the *decoder*, or the **front-end** and the **back-end** of the autoencoder.

But shrinking or increasing the code layer size does not always suffice and other constraints have to be applied on the autoencoder in order to learn meaningful features.

With **denoising autoencoders**, inputs are corrupted with some random noise and the architecture is trained to learn their denoised version. The autoencoder is therefore forced to learn good features of the input but, as well to delete the noise added to this latter.

Sparse autoencoders are architectures where a **sparsity constraint** has been applied on the latent representation. **Sparsity** in the code layer implies

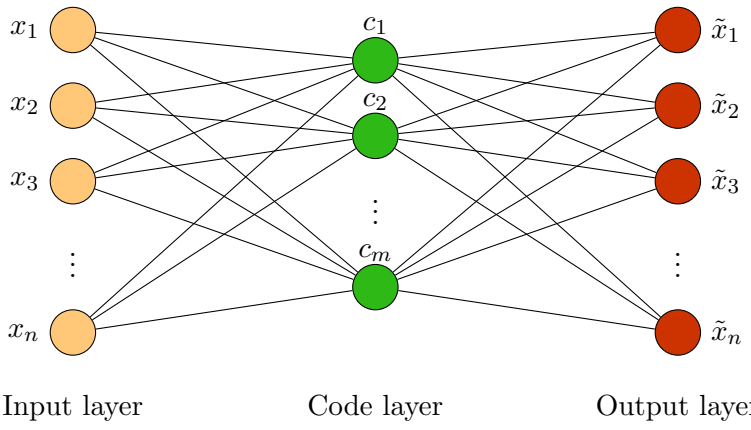


Figure 2.8: Example of a one layer fully connected autoencoder. Here the coding layer has a smaller dimension than the input dimension.

that many units will be very close to the zero value and therefore units will be activated just to extract a very useful information for the reconstruction. For instance, an overcomplete autoencoder ($dim(c) > dim(x)$) without any sparsity constraint will be likely to learn similar features over the code layer or to converge to the identity function.

This constraint is added by forcing the latent units to have a small value ρ in average. The activation value average of a code layer neuron over n samples $(x^{(1)}, \ldots, x^{(n)})$ can be computed by:

$$\hat{\rho}_j = \frac{1}{n} \sum_{i=1}^{n} c_j(x^{(i)})$$

Then, the sparsity constraint consist in imposing:

$$\forall j, \hat{\rho}_j = \rho$$

To do so, a new term is added to the objective function of the autoencoder in order to maintain this equality. Commonly, the Kullback Leibler divergence is used to compare ρ and ρ_j:

$$L_{sparsity}(\theta) = \sum_{j=1}^{dim(c)} D_{KL}(\rho||\hat{\rho}_j) = \sum_{j=1}^{dim(c)} \rho \log \frac{\rho}{\hat{\rho}_j} + (1-\rho) \log \frac{1-\rho}{1-\hat{\rho}_j}$$

The KL divergence can be seen as the measure of lost information when comparing two discrete probabilities, here every ρ_j are approximating ρ.

In our study, the Adaptive layer presented in section 4.1 is based on a sparse autoencoder structure.

2.3.5 Loss function and Backpropagation

In order to train a neural network to learn a specific function in order to realize a specific task, it is needed to tell to the network '*how much it is wrong*' in order to correct its next prediction. This is realized using an **objective function/loss function/cost function** which is telling the goal of our problem. This task can be seen as an optimization problem where the goal is to minimize or maximize the objective function in order to reach its global minimum or maximum point (depending on its definition) and thus reach the optimal solution. In order to reach this optimal goal associated with a particular objective with a deep neural networks the backpropagation algorithm coupled with the stochastic gradient descent algorithm.

The **backpropagation algorithm** [38] is the method used to compute the partial derivatives needed for the stochastic gradient descent algorithm. It is commonly said that the backpropagation step encompasses the gradient computation and the weights update of the neural network,but actually there are three distinct steps, (1) the forward step consisting in inputting the training samples and computing the neural network outputs, (2) the backpropagation computing the partial derivatives of the objective function and finally (3) the stochastic gradient descent algorithm is applied and all the weights are updated.

The backpropagation algorithm is using the **chain rule** in order to compute the gradient of the objective function $L(\theta; X)$ with respect to each weight θ_i (their

influence on the loss function) defined as $\frac{\partial L(\theta;X)}{\partial \theta_i}$. For instance, if $L(\theta; X) = h(\theta)$, using the chain rule, we can compute $\frac{\partial L(\theta)}{\partial \theta}$ as:

$$\frac{\partial L(\theta)}{\partial \theta} = \frac{\partial L}{\partial h}\frac{\partial h}{\partial \theta}$$

If the whole network is made of differentiable operations then it is possible to *backpropagate* the gradient all along until the input layer and thus compute the influence of each weight and update them accordingly using the stochastic gradient descent formula:

$$\theta^t = \theta^{t-1} - \alpha \frac{\partial L(\theta^{t-1})}{\partial \theta}$$

The vanishing gradient problem by the sigmoid function mentioned in subsection 2.3.1 is explained by the fact that during the training phase, bacpropagation algorithm is used and the minimum (which can be local) is searched iteratively in the opposite direction of the error derivative. During the backpropagation process, the sigmoid derivative function is used and multipled between each layers. The problem is that this function lies in $(0, 1/4]$ and these multiplications make the gradient vanish the more it backpropagates it. Furthermore, because of this problem, the first layers of our Neural Network are slower to train than the last one and this might results in a huge inaccuracy at the end. For example in CNNs, as mentioned, first layers learn large features so that the next layers refined these features as much as it goes deeper. The vanishing gradient problem implies that the refined features will have in input corrupted information from the previous layers and therefore will not be able to learn useful patterns.

2.3.6 Regularization

One of the biggest problem that neural networks can encounter during its training phase is **overfitting**, meaning that it can predict very well on the training data but cannot generalize to new data (Figure 2.9). Regularization tries to solve this overfitting problem by penalizing the objective function with a certain additional term.

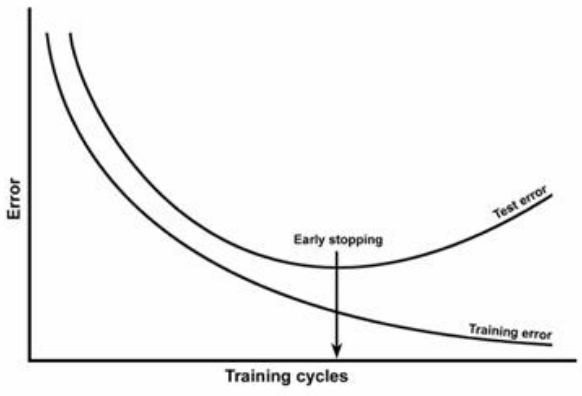


Figure 2.9: We say there the network is **overfitting** the training set when the training error continues to decrease while the test error is increasing - Here the figure explains the concept of **early stopping** to counter the overfitting problem

$$L_{reg}(\theta; X, y) = L(\theta; X, y) + \lambda\Omega(\theta)$$

Here, λ is named the regularization coefficient and $\Omega(\theta)$ is some function contibuting positively on the objective function in order to penalize it with the parameters θ. In this section we present several regularization functions and methods to decrease the generalization error.

The L_2 **regularization**, also known as the ridge regression in linear regression, constraints the network to be trained on small weights, and is defined as follow:

$$\Omega(\theta) = \frac{1}{2}||\omega||_2^2$$

The L_1 **regularization**, also known as the LASSO penalisation, tends to produce sparse weights in the neural network, meaning that many of them will be very close to the zero value:

$$\Omega(\theta) = ||\omega||_1 = \sum_i |\omega_i|$$

Dropout [44] regularization method consists in removing randomly a hidden or visible unit in a neural network every step with the probability p. Dropping out units is done independently in each layer and at each step. This method can be seen as an *ensemble learning* where each dropped out network learns data in its own way, with different errors, and in that way, by combining them averagely, this leads to a stronger architecture and avoids overfitting. At the test time the whole neurons of the network are used.

Another possible regularization during the training phase is to **increase the number of samples** in the dataset used. This can be done by generating new samples from the current dataset with some operations like flipping, rotating, clipping or adding noise.

One of the mostly commonly used and simple technique is called the **early stopping** (Explained in Figure 2.9). This approach consists in evaluating the network on a validation set during the training phase after a certain number of step, e.g. after every epoch. Overfitting can therefore be detected if the current validation set evaluation is higher than the previous one. Commonly, it is set to stop the training after a certain number of evaluations with a lower accuracy than the previous best one.

In this study, early stopping is used to avoid overfitting, we set it so that the training phase stops when the validation set accuracy does not decrease anymore after 5 evaluations. In addition, recurrent dropout is used in the stacked BLSTM networks.

2.3.7 Optimization

During the optimization phase, three types of gradient descent exists. Firstly, the **batch gradient descent** is taking in input the whole dataset and then updating the weights accordindly. Using this method guarantees to converge to the global minimum for convex objective function and local minimum for non-convex one but this method is intractable for huge dataset which can not fit entirely in memory and is not able to handle online learning. The second method, named the **stochastic gradient descent** (SGD), consists in updating the weights after each sample has been fed to the network and is therefore way faster than the first one and can use online learning training. But there can be a huge variance between each update and thus the training phase becomes very noisy in terms of convergence. The third and today commonly used method is the **mini-batch**

gradient descent which considers n samples for updating the weights. Thus, the variance between each update is decreasing as much as the minibatch size increasesn but this size becomes a new hyperparameter to tune for training a neural network. This latter method is often called directly the SGD by misuse of language because minibatches are now universally used during the training phase.

In this section we present three main extensions for the Stochastic Gradient Descent: the momentum, RMSProp optimizer and Adam optimizer.

Momentum

One problem with the SGD method are ravines, surfaces where the curve is steeper in a direction than the other. Since SGD is pointing to the steepest direction rather than in the direction of the optimum, it will oscillate before reaching the surface minimum. Adding momentum helps the SGD to accelerate in ravine surfaces and is defined as follow:

$$v_t = \gamma v_{t-1} + \eta \nabla_\theta J(\theta)$$

$$\theta_t + 1 = \theta_t - v_t$$

The momentum term γ is commonly set to the value 0.9. One extension to the Momentum method is the Nestorov Accelerated Gradient (NAG) [cite].

RMSProp optimizer

Root Mean Square Propagation (RMSProp) optimizer [20] consists in exponentially decaying the learning rate using the average of squared gradients. Defining $g_t = \nabla_{\theta_t} J(\theta_t)$ gradient of the loss function to the parameter θ_tt, η the initial learning rate, RMSProp method is defined as follow:

$$E[g^2]_t = 0.9E[g^2]_{t-1} + 0.1g_t^2$$

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{E[g^2] + \epsilon}} g_t$$

Adam optimizer

Adaptive Moment Estimation (Adam) optimizer [26] approach can be seen as an update of RMSProp where in addition to computing exponentially decaying squared gradient v_t, it considers as well the exponentially decaying gradients m_t as follow:

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t$$

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2$$

Since it has been observed that m_t and v_t are biased for β values close to 1, a bias correction is applied and:

$$\hat{m}_t = \frac{m_t}{1 - \beta_1}$$

$$\hat{v}_t = \frac{v_t}{1 - \beta_2}$$

Finally, we can see Adam optimization method as a combination of RMSProp and Momentum defined as:

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon}\hat{m}_t$$

It has been shown (Figure 2.10) that Adam can lead to better results in terms of convergence than other optimizers.

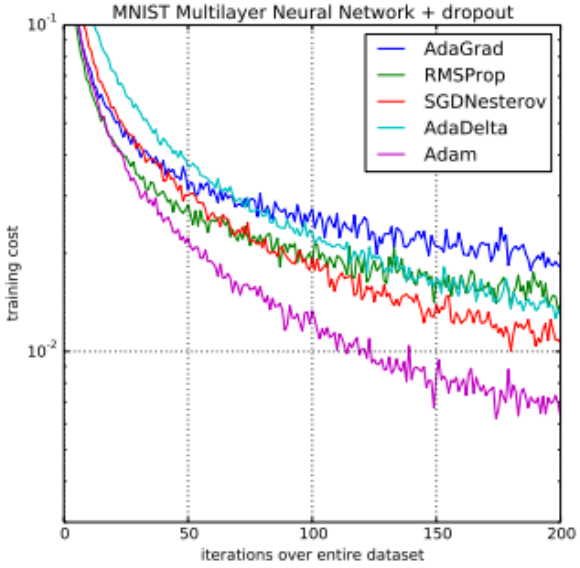


Figure 2.10: Adam Optimizer has better convergence than other optimizers such as RMSProp on the MNIST dataset with a multilayer neural network with dropout - Figure from Adam's paper [26]

In our study, we use both RMSProp and Adam optimizers and select the one leading to the best accuracy. To do so we run the same experiments with different optimizers and learning rates.

Chapter 3

Related work

In this chapter, we firstly discuss about the previous methods used to perform the multi speaker separation problem and secondly, since our study is mainly based on these works, we explain in more details both versions of Deep Clustering, the Source Contrastive Estimation and the Deep Attractor Network methods.

Firstly, [54] and [53] are one of the very first works from M. Weintraub trying to separate two different speakers using Markov Models to infer binary masks. He introduced the GRASP (Grouping Research on Auditory Sound Processing) method which uses onset and pitch features of speeches in order to extract useful information about the speakers and being able to separate them. Following this work, Computational Auditory Scene Analysis systems (CASA) [8, 10] were introduced. They are systems extracting sounds components and are based on the biological inspiration from the Auditory Scene Analysis (ASA) [6]. They try to reproduce source organization achieved by human beings using two main steps: the segmentation phase and grouping phase. Firstly, the segmentation phase extracts information such as harmonicity, pitch, coherent amplitude, or onsets, and then the grouping phase clusters features having similar characteristics to achieve the separation. But since these systems are mostly hand-crafted for the feature extraction they can not perform well on very noisy environments and therefore on real world application.

Factorization models such as the Non Negative Matrix Factorization [39, 29] (NMF or NNMF) consist in learning in a supervised or unsupervised manner a good factorization of a learned dictionnary and activatons bases from spectrograms magnitudes. Then the learned bases can be used to separate the speaker voice they are associated to. But NMF methods are very costly and generate weak representation for the source separation problem. Moreover, NMF struggles to generalize to new environments and voices, it is more successful in structured signals like in music. All theses previous methods did not have very good performances for the multi speaker separation problem because of they struggle generalizing to other speakers.

With its success in many areas like image processing, Deep Learning has recently been as well intensively applied in the audio processing domain. For instance, architectures such as WaveNet [46] and more recently Tacotron 2 [40] have brought a huge leap in terms of performances compared to previous techniques in speech synthesis from text.

In the area of speech enhancement, which is closely related to source separation, fully connected networks [56], stacked LSTM layers [55, 2], deep stacked autoencoders [30] and more recently residual networks (ResNet) coupled with visual information [12] significantly improved the results in this domain in terms of Signal to Noise Ratio (SNR).

For the source separation problem, the first works involving deep neural networks tried to directly infer binary or soft masks using fully connected networks [16, 41, 50, 15], CNN [18, 43] and later networks such as stacked recurrent neural networks [51, 23] and LSTM [49, 25, 12] were used to perform this task. In [14, 34] a unique denoising autoencoder per source was trained to extract especially this latter. In addition, applying clustering algorithm on the latent representation of an autoencoder trained on spectrograms magnitudes of speakers mixtures did not lead to a good separation [4]. All these first methods have a drawback that is a bad generalization since they are all class-based methods and therefore are good on the targets they were trained on.

In the task of voice and music separation, [43] is one of the first work infering masks to separate music and lyrics with CNN layers, later, [31] using the Deep Clustering method coupled with direct mask inference significantly improved the results in this area.

Today's methods performing well in terms of SDR improvement for the single channel multi-speaker separation problem are deep learning architectures outputting discriminative embeddings for each TF-bins in order to be able to separate all the sources. Deep Clustering [19] is the first work introducing this method for the source separation problem, other other works such as Deep Attractor Network (DANet) [7] and Source Contrastive Estimation (SCE) [45] followed this approach. In parallel, Permutation Invariant Training (PIT) [27, 58, 57] method used first CNN and now BLSTM layers to infer soft mask, compared to previous works directly inferring masks with such deep neural architectures, PIT is able to solve the permutation by considering all the possible permutation in output and is trained on the most probable match.

Other works on the source separation problem combines the use of audio mixtures and videos [2, 11, 12], and show that this additional visual information can lead to similar and even better results than previous methods. In particular, [11] shows that the deep neural networks coupled with videos are mostly focusing their attention on the speaker mouth to identify who is speaking.

In [32, 47], end-to-end non-linear architectures that are based on the autoencoder principle are applied directly on audio signals without using the STFT operation as preprocessing. The Adaptive layer presented in Section 4.1 is mainly based on [47] that shows that an autoencoder architecture can improve the source separation performance compared to the use of the audio mixtures Dicrete Cosine Transform as input.

3.1 Deep Clustering (DPCL)

Deep Clustering [19, 24, 52, 31] is one of the first method with a great accuracy on speech separation using deep learning architectures. The significant improvement of deep clustering compared to previous methods is that this approach is a partition-based segmentation algorithm in opposition to the previous class-based methods. Therefore, this method is really good at generalizing to other speakers where previous ones were struggling because only capable to decently handle examples there were trained on. The flexibility and good generalization brought by Deep Clustering is one of the reason of the leap it made in terms of SDR improvement compared to previous methods.

In this section, we explain in details how Deep Clustering can efficiently separate voices from a mixture and in a second phase we introduce its second version enhancing the results using some new methods on which our study will be in part based on.

3.1.1 Discriminative Embeddings

Deep Clustering is using the spectrogram magnitude These spectrograms are fed into a deep neural network that is outputting embeddings $V = f_\theta(x) \in \mathbb{R}^{T \times F \times E}$. With E the size of the embedded space. Beside theses embeddings, a target partition matrix $Y \in \mathbb{N}^{T \times F \times C}$ is created, with C designating the number of speaker in the input mixture. This matrix is defined so that $y(t,f)^{(c)} = 1$ if the speaker i is dominant at the (t,f) bin, $y(t,f)^{(c)} = 0$ otherwise, corresponding to the Ideal Binary Mask of the mixture. The affinity matrix $A = YY^T$ is then created and the estimated affinity matrix can be constructed with the inferred embeddings V with $\hat{A} = VV^T$. The network is then trained such that A equals $\hat{A}$ with the following cost function:

$$L_{DPCL}(\theta) = \left\| A - \hat{A} \right\|_F^2 = \left\| YY^T - VV^T \right\|_F^2$$

The deep neural architecture used in this study is a two stacked BLSTM of 600 hidden unit (300 units in each LSTM cell) with a feedforward neural network projecting the output in the embedded space of size E. This neural network is trained on the objective defined previously with spectrogram mixtures and the ideal binary mask used to construct the affinity matrix YY^T. Each generated embeddings by the deep neural architecture is unit normed.

During the inference phase, Deep Clustering method can solve the permutation problem using three different methods. The first one consists in giving in input the whole utterance to separate each source, in this case, k-means is used on the generated embeddings to create the partitions corresponding to each speaker. Secondly, instead of giving the whole mixture, like in the training, the embeddings are inferred on utterance chunks and k-means is applied on each chunk. But a permutation problem has to be solved in order to assure a continuity between each chunk, and to do so they compute the permutation giving the lowest L^2 distance between the whole separated spectrograms and the original one. In their paper, both method could deliver

3.1.2 End-to-end architecture

As explained, the first version of Deep Clustering [19] approach constructs binary masks via the k-means algrithm applied on the embeddings to perform source separation. In their second paper [24], they introduced an additional layer to transform these binary masks into soft masks and applied a soft differentiable version of k-means in order to finetune the whole network. The final network consists of the deep clustering part, the soft k-means and finally the enhancement layer. Furthermore, they improved their architecture by adding regularization such as dropout and gradient normalization during the training phase. They have shown as well that using more stacked BLSTM with a smaller number of units inside each LSTM leads to better results. In addition, they **pretrain** their network on small context window of size 100 and then finetuned it with a context window of 400. Finally, an end-to-end training was applied to finetune the whole network and led the best result they could obtain.

The enhancement layer consists in concatenating the separated spectrogram magnitudes with the spectrogram magnitude of the mixture along the frequency axis: $Y_j = [\tilde{X}_j, X] \in \mathbb{R}^{T \times 2F}, j \in [1 \dots M]$. This tensor is fed to a BLSTM network followed by a feedforward neural network projecting the output on $z \in \mathbb{R}^{S \times TF}$. This output represents the new assignement for each separated speaker.

A softmax function is then applied in order to output new soft masks $\tilde{m}_i$ in $[0, 1]$:

$$\tilde{m}_i = softmax(z)_i = \frac{e^{z_i}}{\sum_j e^{z_j}}$$

Then these masks are applied on the mixture spectrogram to separate each source:

$$x_i = m_i \cdot X$$

TODO : Loss

We implement this layer to first reproduce the results of DPCL [24] and secondly to apply it to the SCE method in order to enhance its results.

3.2 Source Contrastive Estimation (SCE)

The Source Contrastive Estimation method (SCE) [45] is based on the same approach than Deep Clustering consisting in generating contrastive embeddings in order to apply clustering, create masks and separate all the sources. This method is based on the *word2vec* work. Instead of learning to approximate the affinity matrix $A = YY^T$, the Source Contrastive Method applies a stronger constraint on the generated embeddings. **TODO : details**

$$L(\theta) = -\frac{1}{M} \sum_{t,f} \sum_{s} \log \sigma(Y_{t,f}^{(s)} \cdot v_i(t,f)^T v_o^{(s)})$$

This loss function tends to pull together embeddings and speaker vector from the same speaker and on the contrary tends to push away the one not belonging to a specific speaker.

3.3 Deep Attractor Network (DANet)

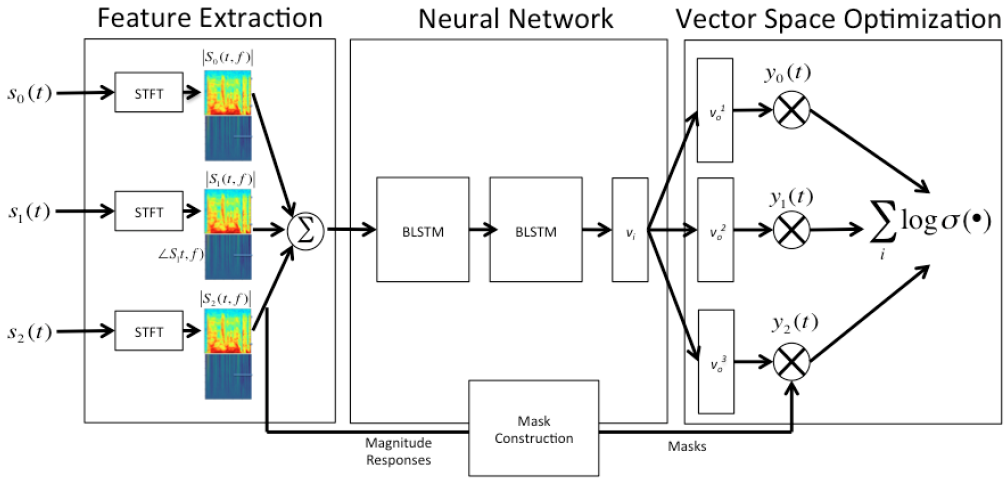
Deep Attractor Network (DANet) [7] method has been proposed after Deep Clustering and and is using the same atchitecture than this latter. Compared to SCE and Deep Clustering, DANet can directly infers the separation masks from the outputted discriminative embeddings. To do so, DANet creates what the authors are calling **attractors**, they are the mean vectors of the embeddings belonging to each speaker, this embedded vectors can be seen as a speaker identifier vector and are defined as:

$$a_s = \frac{\sum_{t,f} Y_{(t,f)}^{(s)} V_{t,f}}{\sum_{t,f} Y_{(t,f)}^{(s)}} = \frac{Y_s^T V}{\sum_{t,f} Y_{(t,f)}^{(s)}}$$

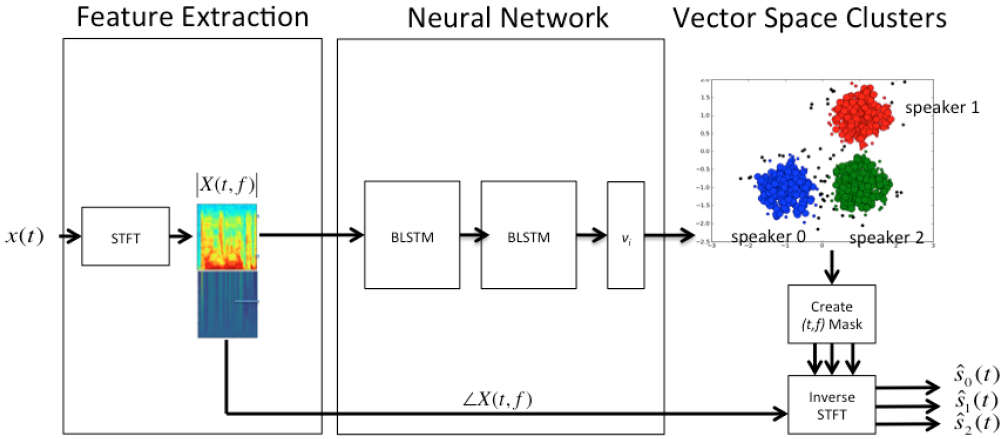
Here, Y_s represents the binary mask extraction the s^{th} speaker and V is the dicriminative embeddings outputted by the deep neural network. Then, masks are computed using the inner product between these attractors a_s and the outputted embeddings d_s for each speaker:

$$d_s = a_s V, s = 1, ..., M$$

Finally, to project these assignements into the $[0, 1]$ interval, the *sigmoid* or $softmax$ functions, defined here as $f()$, are applied on d_s:



(a) Training phase of SCE



(b) Inference phase of SCE

Figure 3.1: Architecture of the Source Contrastive Estimation method. In Figure 3.1a is described the training phase: it consists in producing discriminative embeddings from a mixture spectrogram and then pulling together embeddings from a specific speaker or pushing away embeddings from a different speaker via the loss function defined in this section. In Figure 3.1b, the inference phase consists in producing embeddings and applyting a clustering algorithm such as k-means in order to produce masks and separate each source from the mixture - the figures are from [45]

$$\tilde{m_s} = f(d_s) \in [0, \dots, 1]$$

Once masks are inferred, these latter are compared to ideal binary masks using the L_2 loss:

$$L_\theta(x) = \sum_{i=1}^{M} \|X \cdot (m_j - \tilde{m_j})\|_2^2$$

During the training phase, the network is taught to produce the best reconstruction possible via binary masks.

During the inference phase, since the assignements of each TF-bin is unknown, attractors are computed using K-means algorithm. In DANet paper, it is shown that using K-means is leading to the best results. The authors compared this

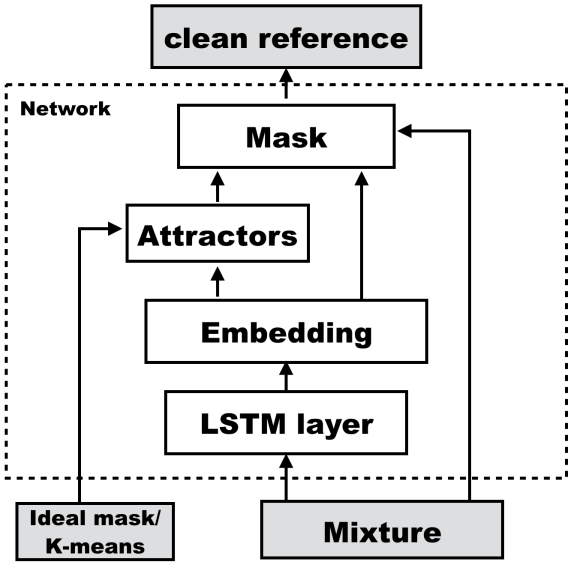


Figure 3.2: Deep Attractor Network (DANet) organisation. During the training phase DANet use the ideal masks produced from the mixture to compute the attractors and then compute the associated masks to separate each source and apply its loss function. During the inference phase, DANet uses the K-means algorithm to cluster the embedded space and construct the attractors - Figure from [7]

approach with the use of fixed attractors computed from a test set, but this method is limitations the generalization efficiency of the network since these fixed attractors are based on known data and therefore make the method less flexible.

In section 4.3, we add DANet loss function to the SCE approach as a regularization for the training phase.

Chapter 4

Contributions

In this chapter we present in more detail the contributions mentioned in 1.2. Firstly, we describe the architecture we use to replace the use of spectrogram magnitude as inputs and that we use with Deep Clustering and Source Contrastive Estimation. Secondly, we describe the improvements we propose for the SCE method, such as using a soft version of k-means, negative sampling, a silent loss function and other approaches brought by [24], the second version of Deep Clustering.

4.1 The Adaptive Layer

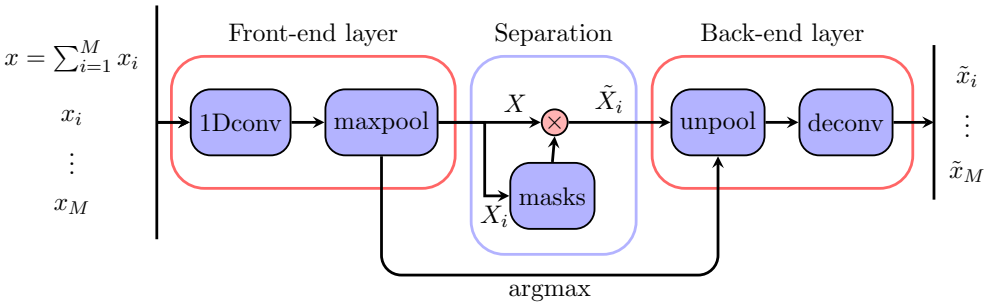


Figure 4.1: Architecture of the Adaptive layer during the **pretraining phase**. The inputs are the audio mixture concatenated with each original speech. These latter are fed to the Front-end and masks are computed from the original speech latent representation. Finally, these masks are applied on the mixture latent representation and reconstructed via the back-end layer.

The main idea in this section is based on an end-to-end architecture presented in [47]. This work shows that the use of a non linear autoencoder directly on audio mixture can lead to better performances than using the DCT of these latter. This autoencoder consists in two main parts, as explained in Section 2.3.4, a front-end creating a latent representation of the input and a back-end reconstruction the input.

Our idea is to apply such an approach to the current deep learning state of the art methods - in this study DPCL and SCE - in order to see if this method can lead to any improvement.

First of all, since Deep Clustering and Source Constrastive Estimation methods are not trained in an end-to-end way, meaning that the output of the network during the training phase does not lead to the reconstruction of the separated signals, then it is impossible to simply add such an architecture '*around*' the

deep neural architectures and train it as a whole. In fact, if we want to train one of these methods using an autoencoder, then only its front-end part can be plug on since the network does not output separated inputs and therefore the autoencoder can not be trained during the training phase of these two methods. Moreover, if only the front-end is used during the training phase then the latent representation will not have any sense in terms of feature extraction since the reconstruction phase is absent.

The solution we propose consists in pretraining the autoencoder (Figure 4.1 for the multi-speaker separation problem. The main difference with standard autoencoders is that in this case we do not train to reconstruct the input but to reconstruct the separated audio speeches. To do so, the autoencoder is **(1)** fed with a **mixture of signals and the original signals from this mixture**, then **(2)** computes the separation and **(3)** reconstructs each separated signals.

To perform these three operations, we present a **sparse linear autoencoder** consisting of a front-end layer, a separation operation and a back-end layer. The front-end layer is a one-dimensional convolution operation and a max-pooling layer and the back-end layer is composed by an unpooling and a deconvolution layer.

Front-end layer As mentioned, the input consists in a audio mixture $x = \sum_{j=1}^{M} x_j \in \mathbb{R}^L$ with the original signals $x_j \in \mathbb{R}^L$, M being the number of mixed signals. We name X the output of the front-end layer corresponding to x and X_j the outputs corresponding to the original signals x_j. Since we want to perform a quasi perfect separation between the encoder and the decoder, the front-end layer must be as linear as possible. Indeed, if the linearity is lost by this latter $X \simeq \sum_{j=1}^{M} X_j$ does not hold and therefore it is not possible to apply filters in order to separate each signal. The front-end we define is composed of two operations. The first one is a **one-dimensional convolution operation** applied on the inputs as follow:

$$Y = 1DConv(x) = W_f * x = (|\omega| \cdot B) * x$$

With $W_f \in \mathbb{R}^{F \times N}$ the F convolution filters of size N applied along the time axis. Here, this matrix is more specifically defined as $W_f = |\omega| \cdot B$ with $\omega \in \mathbb{R}^N$ representing the window whereas $B \in \mathbb{R}^{F \times N}$ represents the bases of the operation.

This convolution layer is followed by a **maxpooling operation** along the time axis with a stride and hop of size m:

$$X = maxpool(Y, m)$$

This maxpooling layer is a non-linear operation but for $M = 2$, we can suppose that $X \simeq \sum_{j=1}^{M} X_j$. We will see in Section 4.1 that the maxpooling operation in the Adaptive layer is indeed a drawback for the reconstruction for $M > 2$ but that this operation is necessary to have a decent sparsity in the latent representation.

Separation: Once the encoder outputs are computed it is possible to separate the mixture representation using the original separated signals representation. For instance, m_j is the mask extracting the j^{th} original signal and is computed as:

$$m_j = \frac{X_j}{\sum_j X_j + \epsilon} \simeq \frac{X_j}{X + \epsilon}$$

The it suffices to apply these masks on the mixture representation to have the separated audio speeches:

$$\tilde{X}_j = m_j \cdot X$$

Back-end layer: The decoder is processing the separated latent representations to reconstruct the original signals. First, it applies an unpooling and then a deconvolution operation. The unpooling operation we use is following the method proposed in [59] that is putting the values where the maximum values were computed by the maxpooling of the front-end, with zeros anywhere else. This method presents better results than gathering the values at the up-left corner of each unpooled patches because it is structurally preserving information. Then, the deconvolution layer has the structure as the front-end convolution but does not share the same weights. We name the reconstructed separated signals $\tilde{x}_j$

Objective function: To train our autoencoder we are using the Mean Square Error (MSE) loss with a sparsity constraint and an additional loss that we name the *overlapping* loss:

$$L_\theta(x) = \frac{1}{M}\sum_{j=1}^{M} \|x_j - \tilde{x}_j\|^2 + \beta KL(\rho, \hat{\rho}) + \alpha \cdot overlapping(x)$$

The overlapping loss function is added in order to reduce the overlapped number of information and therefore push out model to learn good bases functions to distinguish well each signals. This loss is comparing each couples of the sparse representations generated by the front-end layer (without considering the one involving the mixture of all the signals). For a couple (X_1, X_2)

$$overlapping(X1, X2) =$$

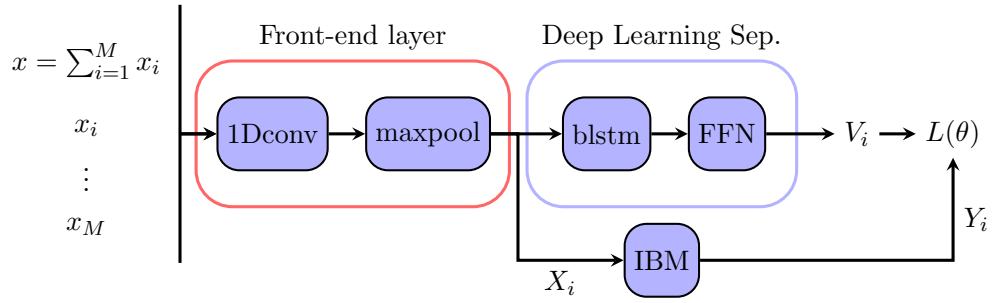


Figure 4.2: Architecture of the Adaptive Layer during the training phase of Deep Learning separation architectures. The inputs are the audio mixture concatenated with each original speech. These latter are fed to the Front-end and ideal binary masks are computed from the original speech latent representations. Finally, the computed latent representation of the mixture is fed as input of the deep neural network, such as Deep Clustering or SCE, and the loss function is computed to update the separator weight using the backpropagation algorithm.

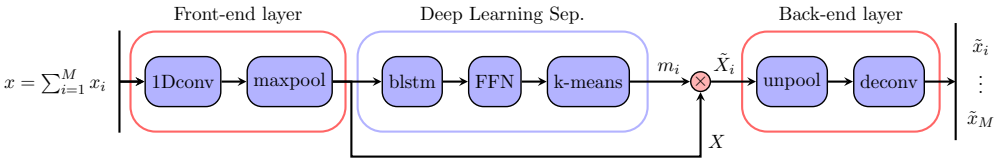


Figure 4.3: Architecture of the Adaptive layer during the finetuning phase.

In Section 4.1, we present the results with different hyperparameters for the pretraining phase of the Adaptive layer. Then we apply the pretrained front-end to DPCL and SCE to train these deep neural networks and we finally plug the back-end layer to finetune the whole network. But to be able to finetune the whole network, the clustering operation must be differentiable.

4.2 Soft K-Means implementation

As explained in Section 3.1.2, the second version Deep Clustering used a soft version of k-means in order to be able to apply backpropagation on the whole network and therefore being able to finetune it in an end-to-end way. In this section we will explain what this soft k-means algorithm consists in and how we implemented it efficiently on the TensorFlow framework.

The idea of using a soft version of k-means is to first be able to apply back-propagation on the whole network since the $argmax$ function used in the hard version is not differentiable and secondly, to produce softer masks and thus improve the separation.

Let's consider the separation of L points $x \in \mathbb{R}^E$ in K clusters. Firstly, it consists in computing the soft assignment γ of each point of the dataset to the centroids $\mu \in \mathbb{R}^{K \times E}$ that are randomly initialized. The assignment of the i^{th} element to the c^{th} cluster is defined as:

$$\gamma_{i,c} = \frac{e^{-\beta|v_i - \mu_c|^2}}{\sum_{c'} e^{-\beta|v_i - \mu'_c|^2}}$$

Then each centroid is updated as being the average sum of the point assignements

$$\mu_c = \frac{\sum_i \gamma_{i,c} w_i v_i}{\sum_i \gamma_{i,c}}$$

This algorithm can actually be interpreted as an Expectation Maximization (EM) with Gaussian Mixture Models (GMM) with a common shared variance. Indeed, a Gaussian Mixture Model is a weighted linear combination of gaussians and is defined as follow:

$$p(x|\mu, \sigma) = \sum_{i=0}^{K} \pi_i N(x, \mu_i, \sigma_i)$$

With:

$$N(x, \mu_i, \sigma_i) = \frac{1}{\sigma_i \sqrt{2\pi}} e^{\frac{(x-\mu_i)^2}{\sigma_i^2}}$$

The first step of the EM method is the expectation step which compute the probabilities for each to point to each gaussian in the mixture, $p(k|x)$ that is the prior probabilities. This step computes what is called the *responsibility*, i.e. how much each gaussian is responsible for each data. The responsabilities correspond to the assignements and are computed using the Bayes rule:

$$\gamma_i(x) = p(i|x) = \frac{p(i)p(x|i)}{p(x)} = \frac{\pi_i N(x, \mu_k, \sigma_k)}{\sum_j \pi_j N(x, \mu_j, \sigma_j)}$$

We can see that if we consider this mixture as being non weighted and having a shared variance then we obtain:

$$\gamma_i(x) = \frac{N(x, \mu_k, \sigma)}{\sum_j N(x, \mu_j, \sigma)} = \frac{e^{\frac{(x-\mu_i)^2}{\sigma^2}}}{\sum_j e^{\frac{(x-\mu_j)^2}{\sigma_i^2}}}$$

Here β is named the *stiffness* and can be seen as the inverse variance of the Gaussian Mixture Models. Therefore, if β is small the gaussian variance will be large and each point will have a higher degree of assignement, in the contrary, for large values the gaussians are narrow and the algorithm is close to its hard version. **TODO Graph with different beta - TODO**

Implementing hard k-means and soft k-means with TensorFlow presented several challenges. First, our version has to be able to handle the whole minibatch of embeddings $V \in R^{B \times TF \times E}$. Indeed, if k-means is applied iteratively on each sample of embeddings this would require a high amount of time and since an unique GPU is used, these operations can not be run in parallel. Therefore, in order to speed up the training time, our implemention is computing hard and soft k-means for a batch of data, at the same time. There was no necessary trick needed for the soft version since assignements and the centroids updates formulas can be extended to batches using tensors multiplications. Whereas with the hard version the assignements designate only one particular cluster, and to compute $\sum_i x[i = c]$ during the centroids update, the function *tf.unsorted_segment_sum* is used to sum the vectors belonging to the same cluster. But , the solution is to shift the cluster assignement for each batch as follow: **TODO**

Secondly, we implemented as well the use of several tries in order to avoid bad centroids initialization and thus improve the accuracy of our algorithm. To do so, we repeat the input T times along its first dimension in order to have $V \in R^{T \times B \times TF \times E}$, i.e. T tries of B batches of TF vectors of dimension E. Seeing these T tries of B batches as TB batches instead reduces this problem to the first one. Once the TB batches are clustered and that the $\mu \in R^{TB \times K \times E}$ centroids are computed by the algorithm, for each batch b the argument with minimum inertia among its tries is selected as the actual clusters to output.

We show in Section 6.4.2 that using soft k-means for the SCE method increases the SDR improvement, but sometimes to the detriment of the SIR improvement.

4.3 Combining DANet and SCE methods

One drawback of the Source Contrastive Loss function is its **locality**, because it is trained on each (t, f) bins without being aware of the global context. For DANet, it is the contrary, it focuses on the globality of the context window by using weighted average vectors of generated discriminative embeddings and do not consider close relation between each embedded vector. Our approach is to consider both methods at the same time in order to apply local loss function with the SCE loss and giving more context by adding the DANet loss function.

The loss function used for such network is therefore:

$$L(\theta) = L_{SCE}(\theta) + L_{DANet}(\theta)$$

$$L(\theta) = -\frac{1}{M} \sum_{t,f} \sum_{s} \log \sigma(Y_{t,f}^{(s)} \cdot v_i(t,f)^T v_o^{(s)}) + \frac{1}{M} \left\| x \cdot (m_i - \tilde{m_j}) \right\|_2^2$$

Once the network is trained on this loss function, we don't use the masks we would get following the DANet method consisting in computing the attractors and then obtaining the masks through $m = sigmoid(d)$. Instead as it is done in

the SCE approach, we compute k-means on the embedded space and then obtain the masks. One extension of this method would be to follow DANet approach to construct the masks.

We show in Section 6.4.4 that this approach is leading to better results for a mixture of 2 speakers of different and same gender.

4.4 Negative Sampling for Source Contrastive Estimation

For the Source Contrastive Estimation method, training to separate two different genders is quite an easy task regarding the speakers embedded vectors since they only have to form two main clusters to correctly learn the difference between male and female speakers. But separating voice from the same gender and two different genders is more challenging since in this case not only two clusters have to be learnt but the number of speakers in the set. For instance, in our case, the LibriSpeech Corpus with 100 hours of speech contains 251 speakers (126 males and 125 females), thus to learn how to separate all possible of mixtures, the training phase has to study the relation between $\begin{pmatrix} 251 \\ 2 \end{pmatrix}$ = couples. This is a very high number of couples to study and it might be impossible to have in input all the possible couples many times, even with a huge dataset generation. As we can see on Figure [CITE], learning with the loss [CITE] on a mixture of 2 speakers with same and different genders leads to a speaker vectors embedded space forming a 'ring'. This repartition leads to areas with bad male/male, female/female contrasts but even male/female in the area between both main male and female clusters.

To tackle this problem, we propose to use *Negative Sampling* method in order to extend the number of couples encountered during the training phase. This method is inspired from NEG objective function defined in [33] where k negative samples are drawn from a certain distribution.

In our case, the general approach consists in selecting, for a (t, f) bin, $v_o^{(k)}, k \in 1, \ldots, K$ speaker vectors different from the dominant speaker vector $v_{o_+}(t, f)$ at this specific bin. In other words, $v_o^{(k)} \in V_o^-(t, f) = V_o/_{v_o^+(t,f)}$, and we define $\mathfrak{B}(t, f) = \{v_o^{(k)}, k \in 1, \ldots, K\}$ the K sampled vectors at a (t, f) bin. We define the new loss function applying Negative Sampling for SCE as follow:

$$L_{NS}(\theta) = L_{SCE}(\theta) - \beta \frac{1}{K} \sum_{t,f} \sum_{v_o^{(s)} \in \mathfrak{B}(t,f)} \log \sigma(-v_i(t, f)^T v_o^{(s)})$$

The first term corresponds to the objective function defined in [CITE] and the second one the added Negative Sampling. In this term $Y_{t,f}^{(s)} = -1$ since we want the K selected vectors to be *pushed away* from the dominant speaker vector, and this is apply for each $v_i(t, f)$ embeddings. The negative sampling term is multiplied by a coefficient $\beta \in [0, 1]$ in order to give priority to the first term since it is the main objective function. Indeed, these two members have both the same magnitude: for instance, choosing $\beta = 1$ would lead to an equivalence in terms of influence for the SCE loss and Negative Sampling loss, and thus, the network would struggle learning the constrast between each speaker vectors because the Negative Sampling term would add too much noisy information to the SCE loss.

In addition, we introduce two approaches to define $\mathfrak{B}(t, f)$. The first one consists in randomly selecting K other speaker vectors from $V_o^-(t, f)$. This ap-

proach is randomly pushing away K other speaker vectors of the embedded space, and therefore will cover all the possible different couples many times during the training phase:

$$\mathfrak{B}(t,f) = \mathfrak{B}_{rand}(t,f) = \{v_o^{(X_k)} \in V_{o_{-1}}(t,f) | X_k \in \mathcal{U}(1,|S|), k \in 1,\dots,K\}$$

The second approach consists in selecting the K-Nearest Neighbors (KNN) vectors of $v_{o_+}(t,f)$ and move them away for it. This idea is based on the local bad contrast obtained between speaker vectors of the same gender and tends to a more spread out speaker vectors embedded space:

$$\mathfrak{B}(t,f) = \mathfrak{B}_{KNN}(t,f) = \{v_o^{(i)} | \left\|v_{(1)} - v_{o+}(t,f)\right\| \leq \dots \leq \left\|v_{(K)} - v_{o+}(t,f)\right\|, i \in 1,\dots,K\}$$

In Section [cite] we show the performances of this approach with different values for K and β. We try this method with spectrograms magnitude and the Adaptive Layer.

4.5 General improvements of Source Contrastive Estimation

4.5.1 Source Contrastive Estimation Silenced Loss

One drawback, we observed with the SCE objective function is that it is taking into account all the TF-bins embeddings $V \in R^{TF \times E}$ outputted by the neural network. The problem is that some of these TF-bins have very low energy and therefore are not very relevant since they don't hold much information about any specific speaker. Considering these almost silent bins is leading to a noisy training because some speaker embeddings will be equally trained on these noisy vectors and on the others holding much more information. Our solution implies to apply a mask on the loss function to omit these almost silent bins, as following:

$$L(\theta) = -\frac{1}{M} \sum_{t,f} \sum_{s} \log \sigma(W_{t,f} Y_{t,f}^{(s)} \cdot v_i(t,f)^T v_o^{(s)})$$

The silent mask W is omitting bins that are higher than a certain level τ of difference in decibel with the highest energy bin of the whole spectrogram:

$$W_{t,f} = \begin{cases} 1 & \text{if } 10\log(\frac{|X_{t,f}|}{max(|X|)}) < \tau \text{ in } dB \\ 0 & \text{else} \end{cases}$$

In Section 6.4.1, we show that using this silent mask is significantly improving SCE results with both spectrograms and the Adaptive layer.

4.5.2 Pretraining, Enhancing and Finetuning

Like in [], we first train the SCE on spectrograms or outputs of the Adaptive front-end with a certain chunk size and then continue to train the network on a bigger chunk size in order to improve its performance. This method can be seen as *finetuning* the network on longer chunk size. Furthermore, being more accurate on longer chunk will with high probabilty improve the performance of our model, indeed during the test phase audio mixtures are chunked with the size the network has been trained on, for small chunk size it is more likely to have a high percenatge of silent area for the speakers and thus a quite noisy sample

with few information to produce a good separation, whereas for longer chunks more information to be able to distinguish them is available. Moreover, after the network has been fully trained on bigger chunks the enhance layer is added to SCE network in order to generate softer and better masks.And finally, since soft k-means is used we can finetune the whole architecture and see if this can lead to any further improvements.

We show in Sections 6.4.5, 6.4.6 and 6.4.7 that these addings are indeed enhancing all methods in terms of SDR and SIR improvements.

Chapter 5

Experiments

5.1 Environment

The framework used for all our deep learning architecture is TensorFlow r1.9 [1] and the experiments were conducted using RAIDEN Computer System provided by RIKEN AIP. This system has nodes with NVIDIA DGX 100 composed of 8 Nvidia Tesla V100-SXM2-16GB of 15GB of RAM. Our experiments were computed on a single GPU but our implementation makes them runnable on multiple GPUs.

To evaluate our experiments in terms of Signal to Distortion Ratio (SDR) and Signal to Interference Ratio (SIR) improvements, we used the BSS_EVAL toolbox [48]. Since we are working on NVIDIA DGX 100, and that these later are shared among RAIDEN users GPU, using the BSS_EVAL Python module on CPU can be very slow when the CPU shared. Therefore, to speed-up the testing phase, we implemented the BSS_EVAL toolbox with CuPy [35] and TensorFlow r1.9 in order to run all the experiments fully on GPUs - note that this new implementation is only compatible with TensorFlow r1.9, which is the first version capable of handling *complex128* numbers for the FFT operation and therefore having good enough precision to deliver similar results as the original CPU-version module

5.2 LibriSpeech ASR Corpus and optimization

For all the experiments conducted in our study we used the LibriSpeech ASR Corpus [36]. From this dataset, we used the *train-clean-100* set containing 100 hours of 125 women and 126 men speeches. This dataset was split in three parts: the training representing 80%, the validation 10% and the test 20% set. Another set was created with *test-other-clean* to evaluate our models on *out-of-set* speaker mixtures. For computational speed up and memory savings we downsampled the audio signals from 16kHz to 8kHz. When spectrograms are used as input, we computed the Short Time Fourier spectral magnitudes for each mixture with a window size of 32 ms (256 samples), a hop size of 8 ms (64 samples) with the square root of the Hann as window. From these spectrograms were build the ideal binary masks Y by evaluating the dominant speaker for each (t, f) bin.

In our implementation, we do not precompute the spectrogram magnitudes and store them like it is usually done. We directly use the raw audio files as inputs because our models have to be able to compare their results on the same audio files using the Adaptive layer or spectrogram magnitudes. To do so, we stored the audio signals in *.tfrecords* files that are processed by the *tf.data.Dataset* API. Using directly the TensorFlow format file and API for data importation enables the use of operations such as *tf.data.map* allowing the manipulation of audio files

on multiple processes and *tf.data.prefetch* which is prefetching next batches on the CPU while the previous one are computed on the GPU. All these optimized operations for the input data generation leads to a significant speed-up for large batches of 256 elements. A good advantage of this method is that it can randomly generate many different mixtures with different chunk sizes, number of speakers and genders without to have to store the inputs on the hard disk - random states are set constant during the training phase in order to have reproducible results. But one drawback is that it is computing exactly the same inputs every epochs and therefore can be seen as quite time consuming. Furthermore, since the non-chunked raw data are stored, our implementation is capable to generate chunked mixtures of different size and therefore can generate. All our code can be found at [9].

Chapter 6

Results

In this chapter we present our results on the environment and dataset presented in [CITE]. In the first section, using the LibriSpeech Corpus, we reproduce the results of the Deep Clustering and Source Constrastive Estimation methods. In the second section, we show our results using the Adaptive layer with these two methods: to do so we first analyze the *pretraining* phase of the Adaptive layer for different hyperparameters and show some drawbacks encountered with this latter. In the third section we try all the contributions mentioned previously and see the improvements we could get.

For all the evaluations shown in this chapter, the models were evaluated on the chunk size they were trained on. To generate our test set we apply the procedure mentioned in [CITE EXP].

6.1 State of the art results reproduction

Improvements in dB	m+m		m+f		all	
	SDR	SIR	SDR	SIR	SDR	SIR
DPCL [19]	4.00		9.07		6.54	
DPCL (ours)	2.71	6.90	8.00	14.91	5.49	9.85
DPCL - enh (ours)	6.79	11.47	10.14	16.09	7.59	12.34
DPCL - ext (ours)	3.21	7.20	10.43	17.97	4.74	9.28
DPCL - ext + enh (ours)	7.92	13.47	12.38	19.18	9.08	14.38
DPCL++ [24]	9.40		12.0		10.08	
DPCL - finetuned (ours)	-	-	-	-	-	-

Table 6.1: Deep Clustering results reproduction

Deep Clustering: In Table 6.1 we report our results for the reproduction of the Deep Clustering approach. We can see that for the DPCL++ [24] approach we could reproduce the results (equivalent to *DPCL - ext + enh (ours)*) but we could not have the same for its first version DPCL [19] (equivalent to *DPCL (ours)*). This might be due to some hyperparameters difference, implementation, training time and type of data.

From now on, we detail the process we followed from reproducing DPCL [19] to DPCL++ [24] results. To reproduce the DPCL results we used an architecture consisting in 4 BLSTM layers of 600 units (corresponding to 300 units in each LSTM cell) with an embedded space of size $E = 40$ - this size is set as constant for all the following experiments. This network is one BLSTM layer larger than the one used in [19], but we see in the Table 6.2 that even using a larger network

DPCL chunk 100	m+m		m+f		all	
k-means β - silence	SDR	SIR	SDR	SIR	SDR	SIR
hard dB	2.85	8.71	8.03	14.80	5.30	11.40
hard - 20 dB	3.58	10.78	8.00	14.91	*5.49*	*12.39*
5 - 20 dB	1.58	2.86	6.22	9.26	2.97	4.49
10 - 20 dB	**2.71**	6.90	**8.42**	14.49	**5.12**	9.85
15 - 20 dB	1.82	7.23	8.34	14.88	4.51	**10.15**
20 - 20 dB	1.60	7.25	8.22	**14.93**	4.25	10.11
25 - 20 dB	1.53	**7.27**	8.13	14.92	4.13	10.08

Table 6.2: Evaluation of a Deep Clustering architecture (4x600) with different hyperparameters for kmeans and different type of mixture with 2 speakers - green cells are the parameters used for the next phase

does not lead to the same results. In addition, instead of using a fully connected network as the final layer of our network, we replaced it by a one dimensional convolutional network, this is often leading to the same results and reduces the number of weights that can be high if the BLSTM units number or the embedded space size increases. Concerning the optimization, we use the RMSProp optimizer (**??**) with a learning rate of $1e^{-3}$ that is halved every 50 epochs. We compare the SDR and SIR improvements using a regular hard version of k-means with and without silence and the soft version with different stiffness β and a threshold of $20dB$. As regularization,as in DPCL++ [24], in each BLSTM is recurrent dropout and a gradient normalization of 200 is used during the optimization phase. These experiments are computed during 100 epochs using early stopping if the validation set accuracy does not decrease after 5 evaluations. In many experiments results, it can be observed that with higher values of β (harder stiffness) for the soft k-means algorithm, the SIR improvement is increasing, this is explain by the fact that the separation is *stronger* (close to binary masks), there is less remaining of other sources in the separated one, and therefore less interferences, what SIR is actually measuring.

DPCL chunk 100 - enhance		m+m		m+f		all	
optimizer - α	dropout	SDR	SIR	SDR	SIR	SDR	SIR
RMSProp $1e^{-3}$	0.0	6.69	11.05	**10.14**	16.09	7.21	11.42
RMSProp $1e^{-3}$	0.2	6.20	10.12	div	div	6.91	10.62
Adam $1e^{-3}$	0.0	6.71	11.45	10.12	16.00	**7.59**	**12.34**
Adam $1e^{-3}$	0.2	**6.79**	**11.47**	10.12	**16.22**	7.29	11.59

Table 6.3: Enhancement of the previously trained Deep Clustering models with chunk size of 100 frames - the enhance layer is evaluated with different optimizers and recurrent dropout values

Then, we plug the enhancement layer presented in **??** in order to produce softer masks and, in doing so, improving the accuracy of our model. In that regard, we use the previous model trained on chunks of size 100 and add the enhancement layer consisting in 3 BLSTM layers of 600 units each. We test the performance of ours models with different optimizers and recurrent dropout values, the results are reported in Table 6.3. The stiffness used for the soft k-means corresponds to the green cells in 6.2, in this study we always select the hyperparameters leading the best SDR improvement,but an interesting extension would be to select the best SIR improvement to analyze if focusing on SIR can

have a bigger impact on the SDR than the contrary.

DPCL chunk 400	m+m		m+f		all	
kmeans β - silence	SDR	SIR	SDR	SIR	SDR	SIR
hard - 0 dB	3.17	10.23	9.86	18.87	4.91	12.25
hard - 20 dB	*4.52*	*13.06*	9.64	18.67	5.53	14.05
5 - 20 dB	1.71	2.19	6.71	8.75	2.28	2.87
10 - 20 dB	**3.21**	7.20	**10.43**	17.97	**4.74**	9.28
15 - 20 dB	1.67	7.87	10.22	18.64	3.64	10.14
20 - 20 dB	1.19	7.92	10.02	**18.66**	3.21	10.22
25 - 20 dB	1.09	**7.99**	9.89	18.63	3.04	**10.27**

Table 6.4: Finetuning the previous model using chunks of size 400 frames and evaluating on different parameters for k-means

Here, we apply the enhancement layer before finetuning our model with bigger chunks to see the difference with applying bigger chunk and then the enhancement layer. Then, we follow [24] and finetune our network of Table 6.2 with chunks of 400 frames during 40000 steps in average. We can see in Table 6.4 that, for a mixture of different gender (m+f), training on larger chunks is already leading to better SDR and SIR improvements than the enhanced on trained on chunks of size 100. But, surprisingly, for the overall mixture set the results slightly decreased in terms of SDR improvements.

DPCL chunk 400 - enhance		m+m		m+f		all	
optimizer - α	dropout	SDR	SIR	SDR	SIR	SDR	SIR
RMSProp $1e^{-3}$	0.0	**7.92**	**13.47**	**12.31**	19.17	9.02	14.33
RMSProp $1e^{-3}$	0.2	7.63	12.81	12.16	19.18	8.78	14.14
Adam $1e^{-3}$	0.0	7.44	12.79	12.28	19.18	**9.08**	14.38
Adam $1e^{-3}$	0.2	7.83	13.30	12.19	**19.36**	9.02	**14.59**

Table 6.5: Enhancing the Deep Clustering models finetuned with chunk of size 400 using Adam or RMSProp optimizer and different reccurent dropout values

As previously, we add the enhancement layer to the models from 6.4 and train the latter on the best SDR improvement reached with the soft-kmeans algorithm - here $\beta = 10$ and a silence threshold of $20dB$ for all set.

DPCL 400 finetuned		m+m		m+f		all	
loss	finetuned part	SDR	SIR	SDR	SIR	SDR	SIR
L_2 loss	whole	-	-	-	-	-	-
	DPCL only	-	-	-	-	-	-
	enhance only	-	-	-	-	-	-
enhance loss	whole	-	-	-	-	-	-
	DPCL only	-	-	-	-	-	-
	enhance only	-	-	-	-	-	-

Table 6.6: Finetuning the whole network

For the final phase, we finetune the whole network with the Deep Clustering part plus the unrolled k-means steps and the enhancement layer. In the DPCL++ paper, the loss function used for the end-to-end finetuning and the finetuned parts are not mentioned. Thus, we try to use the L_2 loss function on the fully

reconstructed signals and the enhancement loss function mentioned in [REF], in addition, we try to finetune the whole network (meaning the DPCL network plus the enhancement layer), only the DPCL network and only the enhancement part. We report our results in the Table 6.6. For the L_2 loss, we apply the masks produced by the enhancement layer on the spectrogram mixture and reconstruct each separated signals using the ISTFT. We can see that ..

Source Contrastive Estimation:

Improvements in dB	m+m		m+f		all	
	SDR	SIR	SDR	SIR	SDR	SIR
SCE [45]	5.48	x	9.98	x	7.69	x
SCE (ours)	3.72	9.75	7.38	14.10	5.89	12.23

Table 6.7: Reproduction of SCE results

In a second phase, we reproduce the results of SCE [45] using the same architecture consisting in 3 BLSTM layers of 600 units each. Before being fed into the network, the square root function is applied on the magnitude spectrograms and these latter are normalized between 0 and 1. We report our results in 6.7: as we can see we could not reach the results presented in [45] but we will see that the addressed contributions are leading to better results for a mixture of 2 speakers.

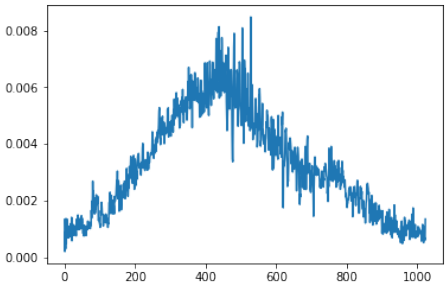
6.2 Adaptive Layer pretraining

In this section we analyze the pretraining phase of the Adaptive layer presented in section 4.1. To do so, we examine the reconstruction performances for different values for the defined hyperparameters.

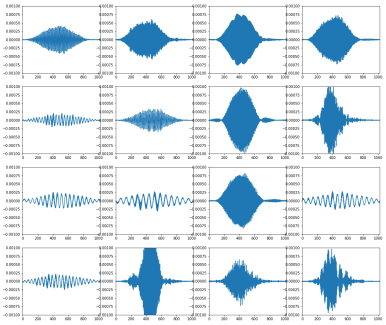
Firstly, concerning the network architecture, the first one dimensional convolutional layer was set to have a stride of 1 and we evaluate the performance for different size of windows and number of filters. The max-pooling layer is set to have the same stride and hop not to have overlapped information and is evaluated with different sizes. Secondly, the RMSProp optimizer was used with a learning rate of 0.001 and was halved each 50 epochs. The overlapping rate λ was set constant equal to 100, the sparsity rate β was set to 0.01 with a sparsity constraint of 0.01. We trained the Adaptive layer during 120 epochs. Concerning the dataset, we used a mixture of 2 speakers with all possible genders combination (male/male, female/female and male/female) to cover as many frequency combinations as possible. Furthermore, the data given in input of the front-end were not normalized and given as raw.

As we can see in the Table [CITE], where different hyperparameters for the Adaptive layer are tried, the best result obtained is for a window of size 1024 and a maxpooling stride and hop of 128. Concerning the number of filters, using 512 filters led to better results but we did not use this pretrained model for our next experiments because having a large number of filters can lead to worse separation performance than using 256 filters. We do not report the results here, but we compared the architectures with 512 filters and the one with 256 filters on the Deep Clustering method and obtained better separation using the latter. For the rest of this study we will use the pretrained model with 256 filters and a window size of 1024 as our Adapt layer for the Deep Clustering and Source Contrastive Estimation methods. In the following paragraphes, we analyze this specific selected model.

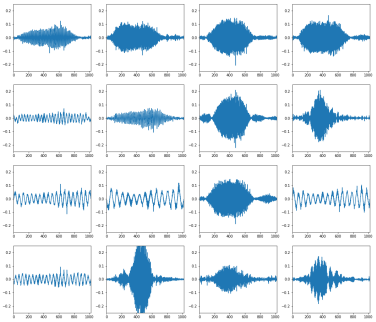
First, in Figure 6.1a we can see that the learnt windows is a bit noisy but



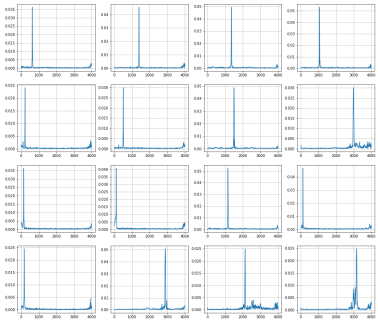
(a) Front-end learnt window $|\omega|$



(b) Front-end learnt convolution filters $|\omega|B_i$



(c) Front-end learnt convolution bases B_i



(d) Fast Fourier Transform of B_i in Figure 6.1c

Figure 6.1: Pretraining results

follows the shape of the usually used windows that are the Hann and Hamming windows. Secondly, in Figures 6.1b, 6.1c and 6.1d, we observe that the bases/filters learnt by the Adaptive layer are extracting frequencies especially for the multi-speaker separation problem. As it could be expected, the filters are more focused on the low frequencies (Figure 6.2)/

In Figure 6.3, we compare the Adaptive layer latent representation with the STFT of the mixture of 2 speakers and their original signals. We can observe that the adaptive layer produces a more compact repsentation than the STFT due to the 128 maxpooling stride and hop. Furthermore, the sparsity of the STFT and Adaptive layer for the original signals appears to be quite the same but the biggest difference lies in the mixture representation where the Adaptive layer is significantly less noisy and way sparser than the STFT representation.

Since the maxpooling layer is leading to reconstruction problems, we as well tried to delete this layer and the unpooling one, and instead of using a stride of 1 from the one dimensional convolutional layer we used larger size of hop (like it is actually done in the Short Time Fourier Transform). We used the same parameters for the overlapping and the sparsity constraint, plus we added a non negativity constraint in the objective function.

As Table [CITE] shows, using the overlapping loss combined with the non-negativity one could lead to better reconstruction. But, we tried to apply these pretrainied autoencoders to SCE and Deep Clustering, and we couldn't reach

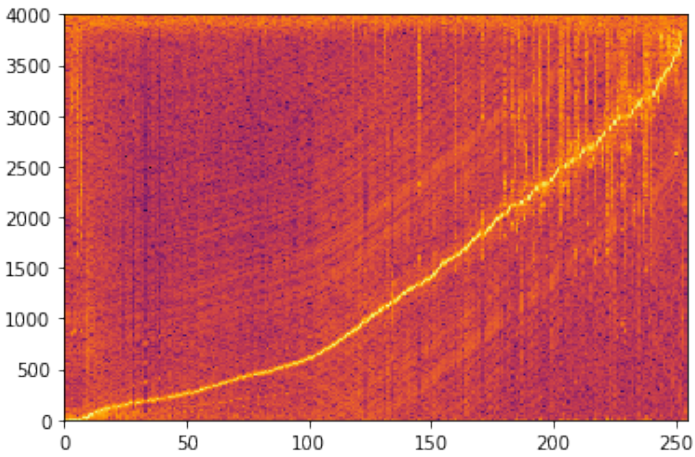


Figure 6.2: FFT sort

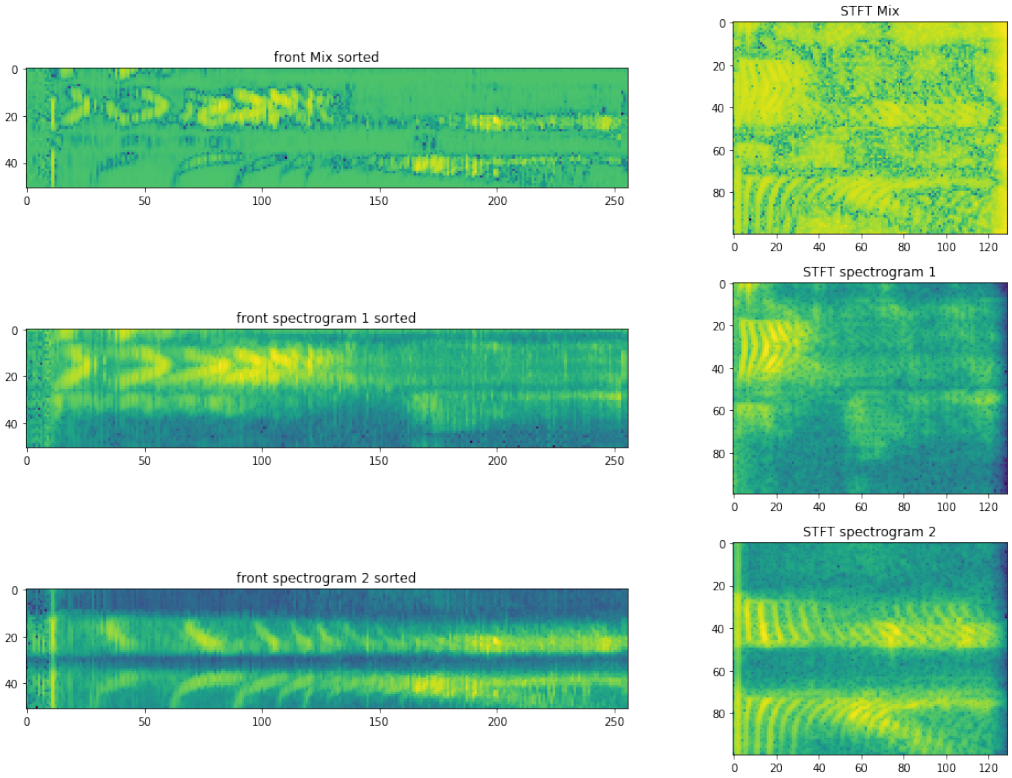


Figure 6.3: Compare

good results. This was mainly due to the bas sparsity of this representation, indeed, as we can in the Figure [CITE], the bases learned by the autoencoder are very noisy and therefore do not extract a specific information. Furthermore, not using the non negativity constraint was leading to very contrasts of sign in the latent representation, and even with it the negativity was l. The problem of having highly negative and positive representation is when, for instance, two spectrograms are added some bins will see their bins changing sign and applying masks will not work on such cases.

The biggest problem for now with the Adaptive Layer is its incapacity to correctly reconstruct signal of more than 2 speakers mixture. As mentioned, this layer is not fully linear, indeed for a mixture of 2 speakers the max-pooling layer

does not have a huge effect on the sparse representation computed but for more speakers it is more likely that the maximum value of the added mixtures has a different position

6.3 Adaptive layer with Deep Clustering

Adaptive DPCL	m+m		m+f		all	
kmeans β - silence	SDR	SIR	SDR	SIR	SDR	SIR
hard	6.36	12.66	10.08	17.12	7.69	14.11
hard - 20 dB	6.58	13.23	10.07	**17.20**	7.68	**14.20**
5 - 20 dB	5.92	9.43	9.71	14.87	6.81	10.19
10 - 20 dB	**6.70**	12.45	**10.26**	17.05	**7.84**	13.64
15 - 20 dB	6.57	12.71	10.17	17.09	7.54	13.82
20 - 20 dB	6.47	12.78	10.09	17.06	7.37	13.79
25 - 20 dB	6.41	**12.81**	10.02	17.01	7.28	13.76
DPCL 100	2.71	6.90	8.42	14.49	5.12	9.85

Table 6.8: Evaluation of the deep clustering method using the Adaptive layer as input with soft and hard k-means - we compare these results to the one obtained using spectrograms (last row)

In this section, we analyze the performances of using the Adaptive layer presented in [REF] instead of spectrogram magnitudes as inputs. To do so, we first only plug the front-end of the Adaptive layer to the deep clustering network]. The outputs of the front-end are normalized with an unit mean and zero variance, but we do not apply functions like the absolute value or logarithm. The input chunk size for the Adaptive front-end is set to 10240 ($\approx$ 1.30 seconds). As we can see in the Table 6.8, using the adaptive layer instead of spectrogram is significantly leading to better SDR and SIR improvements.

Then, in the same way, we finetune the previously trained models with longer audio chunks, here we use chunks size of 30720 frames ($\approx$ 4 seconds), we report our results in the Table 6.10. In this case, we can see that the gap between the previous model and the finetuned one is not as substantial as it is with using spectrograms. For the male+female (m+f) mixture this can be explained by the reconstruction threshold of the adaptive layer.

Adaptive DPCL extended	m+m		m+f		all	
kmeans β - silence	SDR	SIR	SDR	SIR	SDR	SIR
hard - 0 dB	7.79	15.70	10.80	19.73	8.26	16.27
hard - 20 dB	7.58	15.78	10.68	19.65	8.39	16.85
5 - 20 dB	7.62	12.40	10.40	16.45	7.57	11.60
10 - 20 dB	**8.05**	15.41	**10.97**	19.56	**8.50**	15.79
15 - 20 dB	7.91	15.59	10.86	19.65	8.33	16.09
20 - 20 dB	7.80	**15.60**	10.75	19.60	8.21	16.17
DPCL 400	3.21	7.20	10.43	17.97	4.74	9.28

Table 6.9: Finetuning the previously trained model with audio chunks of size 30720 - we compare these results with the one obtained using spectrograms (last row)

Adaptive DPCL ext - enh		m+m		m+f		all	
optimizer - α	dropout	SDR	SIR	SDR	SIR	SDR	SIR
RMSProp $1e^{-3}$	0.0	**8.90**	14.61	11.34	17.78	**9.28**	**14.64**
RMSProp $1e^{-3}$	0.2	8.72	14.55	10.63	16.36	-	-
Adam $1e^{-3}$	0.0	8.12	13.33	10.50	16.61	-	-
Adam $1e^{-3}$	0.2	7.55	**15.67**	10.11	18.89	-	-
DPCL 400 + enh		7.92	13.47	**12.38**	**19.18**	9.08	14.38

Table 6.10: •

6.4 Source Contrastive Estimation improvements

In this section, we analyze the performances of the contributions previously mentioned. All along, we compare the results using spectrograms with the use of the Adaptive Layer. The Adaptive layer is plugged to the SCE network the same way as in [REF]. First we observe the influence of not considering silent bins, different architecture and network regularization. Secondly, we analyze the performances of the soft k-means approach compared to the hard one. In the third section, we apply negative sampling for a mixture of male and female speakers And finally, we apply finetuning to the whole network.

6.4.1 Silent objective function, architecture and regularization

SCE 100	log / meanstd		sqrt / 01	
network / silence	SDR	SIR	SDR	SIR
3x600	7.35	14.07	*7.38*	*14.10*
3x600 20 dB	7.31	14.02	7.36	14.06
3x600 30 dB	7.31	14.02	7.42	14.13
3x600 40 dB	7.32	14.02	**7.43**	**14.16**
4x600 40 dB	7.34	14.02	**7.69**	**14.47**

Table 6.11: Evaluation of the silent objective function for the SCE method with different thresholds, normalizations and architectures - hard k-means is applied to cluster the embeddings and the models are trained on 2 speakers male/female mixtures

Firstly, introduced in [REF], we evaluate the performance of the new objective function not considering silent bins with different thresholds. The results are reported in Table 6.11 using hard k-means. Furthermore, instead of using the square root of the spectrogram magnitude plus a normalization between 0 and 1, we use the log magnitude with a mean and variance normalization, but this approach does not lead to better results. In addition, we add one more BLSTM layer and see that it can lead to better SDR and SIR improvements. For the rest of the experiments using STFT spectrograms, we use a network with 4 BLSTM layers, a silent threshold of $40dB$ and a square root with a 0-1 normalization.

In Table 6.12 recurrent dropout regularization is added to the BLSTM layers, but this does not lead to any improvement in terms of SDR and SIR measures.

SCE 100 40dB	m+m		m+f		all	
dropout	SDR	SIR	SDR	SIR	SDR	SIR
0	**4.29**	**9.75**	**7.69**	**14.47**	**5.89**	**12.23**
0.2	3.34	8.69	6.55	13.17	4.75	10.91

Table 6.12: Evaluation of SCE method with and without recurrent dropout - hard k-means is used here to separation each speaker

Adapt SCE	network	dropout	SDR	SIR
no silence	3x600	0	6.24	11.97
		0.2	5.89	11.57
	4x600	0	6.20	11.81
		0.2	6.04	11.73
20 dB	3x600	0	**7.52**	**13.87**
		0.2	7.00	13.03
	4x600	0	7.03	12.80
		0.2	6.93	12.86
30 dB	3x600	0.0	6.42	12.18
40 dB		0.0	6.19	11.91

Table 6.13: Evaluation of the SCE method using the Adaptive layer with different architectures, loss function and dropout - hard k-means is used to separate each speaker - a mixture of 2 speakers with different gender and a chunk size of 10240 is used. We can see that with the Adaptive layer shorter architecture, no dropout and a threshold silence of 20dB leads to the best results

6.4.2 Soft k-means

6.4.3 Negative sampling

6.4.4 DANet combination

6.4.5 Chunk size extension

6.4.6 Enhancement layer

6.4.7 Finetuning

6.5 Global results

method	m+m		m+f		all	
network	SDR	SIR	SDR	SIR	SDR	SIR
SCE 100	4.29	9.75	**7.99**	**14.51**	**7.41**	**15.06**
Adapt SCE	**4.69**	**10.12**	7.68	13.82	6.05	11.19

Table 6.14: Comparison of the SCE method using spectrograms (first row) and using the Adaptive layer (second row)

SCE 100	m+m		m+f		all	
kmeans β - silence	SDR	SIR	SDR	SIR	SDR	SIR
hard dB	3.72	9.75	7.69	14.47	5.89	12.23
hard - 20 dB	3.63	9.65	7.54	14.24	5.87	12.23
5 - 20 dB	4.04	8.14	7.72	13.22	6.13	10.90
10 - 20 dB	**4.29**	9.75	7.99	14.51	6.38	12.29
15 - 20 dB	4.20	9.96	7.90	14.60	6.29	12.43
20 - 20 dB	4.14	**10.02**	7.85	14.61	6.22	12.46

Table 6.15: lol

Adapt SCE	m+m		m+f		all	
kmeans β - silence	SDR	SIR	SDR	SIR	SDR	SIR
hard	4.35	10.04	7.52	13.87	5.61	11.16
hard - 20 dB	4.39	10.06	7.34	13.68	5.66	11.22
5 - 20 dB	3.39	5.94	7.38	12.34	3.63	6.04
10 - 20 dB	4.58	9.18	**7.68**	13.82	5.50	9.43
15 - 20 dB	4.69	9.86	7.69	13.98	5.97	10.76
20 - 20 dB	**4.69**	**10.12**	7.68	**14.02**	**6.05**	**11.19**

Table 6.16

method	β	K	SDR	SIR
KNN	0.001	5	5.65	11.88
		10	5.60	11.85
		15	5.17	11.25
	0.1	5	5.61	11.79
		10	5.71	11.97
		15	4.20	10.07
	0.5	5	2.06	6.87
		10	1.70	6.25
		15	0.32	3.75
random	0.001	5	5.74	12.06
		10	5.55	11.80
	0.1	5	**6.15**	**12.48**
		10	6.03	12.38
	0.5	5	2.96	8.02
		10	2.38	7.37

(a) Using spectrograms

method	β	K	SDR	SIR
KNN	0.001	5	5.63	10.00
		10	5.63	9.90
		15	5.58	10.09
	0.1	5	5.83	9.97
		10	5.87	10.08
		15	5.95	10.14
	0.5	5	0.22	2.32
		10	0.04	2.28
		15	0.14	2.65
random	0.001	5	5.6	9.73
		10	5.68	9.83
		15	5.61	10.05
		20	5.70	10.14
	0.1	5	6.22	11.17
		10	**6.25**	**11.27**
		15	6.18	11.06
		20	6.19	11.05
	0.5	5	2.17	4.91
		10	2.05	4.56
		15	2.13	4.77
		20	2.06	4.69

(b) Using the Adaptive layer

Table 6.17: Evaluation of the SCE method using spectrograms(6.17a) or the Adaptive Layer(6.17b) with Negative Sampling for a mixture of 2 speakers of same and different genders - random and KNN Negative Sampling are evaluated with different values for K

SCE rand	all	
kmeans β - silence	SDR	SIR
hard	6.15	12.48
hard - 20 dB	6.00	12.33
5 - 20 dB	6.18	10.58
10 - 20 dB	**6.73**	12.54
15 - 20 dB	6.67	12.80
20 - 20 dB	6.60	**12.87**

(a) •

Adapt SCE rand	all	
kmeans β - silence	SDR	SIR
hard	4.19	9.54
hard - 20 dB	3.87	9.65
5 - 20 dB	5.07	8.07
10 - 20 dB	6.25	11.27
15 - 20 dB	6.22	11.83
20 - 20 dB	6.18	12.03

(b) •

Table 6.18: •

method	m+m		m+f		all	
	SDR	SIR	SDR	SIR	SDR	SIR
SCE 100 DANet	5.08	9.74	8.43	14.49	7.24	12.74
Adapt SCE DANet	**5.78**	**11.51**	**9.05**	**15.67**	**7.37**	**13.27**

Table 6.19: •

SCE+DANet 100	m+m		m+f		all	
network	SDR	SIR	SDR	SIR	SDR	SIR
3x600	**3.53**	**6.32**	**8.43**	**14.49**	**k**	**b**
4x600	2.33	4.62	8.30	14.26	6.35	10.67

Table 6.20: •

SCE+DANet 100	m+m		m+f		all	
kmeans β - silence	SDR	SIR	SDR	SIR	SDR	SIR
hard - 0.0	3.90	9.92	7.90	14.29	6.38	12.68
hard - 20 dB	3.52	9.53	7.47	13.94	5.95	12.27
5 - 20 dB	1.43	2.75	7.93	12.75	5.34	8.48
10 - 20 dB	3.53	6.32	**8.43**	14.49		
15 - 20 dB	4.54	8.29	8.33	**14.53**	7.13	12.40
20 - 20 dB	4.97	9.30	8.25	14.49	**7.24**	**12.74**
25 - 20 dB	**5.08**	**9.74**	7.89	14.09	6.86	12.35

Table 6.21: •

Adapt SCE DANet	m+m		m+f		all	
network	SDR	SIR	SDR	SIR	SDR	SIR
3x600	5.15	10.17	8.66	15.02	6.84	11.98
4x600	**5.70**	**10.87**	**9.04**	**15.55**	**7.14**	**12.24**

Table 6.22: Evaluation of the SCE method with the Adaptive layer using different architectures - compared to the one using spectrograms, here using a deeper architecture leads ot better results

Adapt SCE DANet	m+m		m+f		all	
kmeans β - silence	SDR	SIR	SDR	SIR	SDR	SIR
hard - 0.0	5.46	11.30	8.81	15.45	8.38	14.83
hard - 20 dB	5.51	11.33	8.66	15.25	8.38	14.84
5 - 20 dB	4.66	7.93	8.70	14.46	5.18	8.19
10 - 20 dB	5.70	10.87	9.041	15.55	7.14	12.24
15 - 20 dB	5.780	10.38	**9.047**	15.67	7.35	13.04
20 - 20 dB	**5.784**	**11.51**	9.03	**15.69**	**7.37**	**13.27**

Table 6.23: •

method	m+m		m+f		all	
	SDR	SIR	SDR	SIR	SDR	SIR
SCE	-	-	9.98	16.90	7.54	13.17
SCE + NS	x	x	x	x	8.01	15.34
SCE + DANet	**7.12**	**13.42**	**10.36**	**18.33**	**8.58**	**15.38**
Adapt SCE	4.71	10.89	8.28	14.56	6.62	13.16
Adapt SCE + NS	x	x	x	x	6.36	13.22
Adapt SCE + DANet	6.63	13.53	9.84	18.11	7.79	15.16

Table 6.24

SCE chunk 400	m+m		m+f		all	
kmeans β - silence	SDR	SIR	SDR	SIR	SDR	SIR
hard	5.42	12.96	9.57	18.42	7.08	15.0
hard - 20 dB	5.12	12.81	9.10	17.97	6.65	14.66
5 - 20 dB	**5.99**	11.45	**9.98**	16.90	**7.54**	13.17
10 - 20 dB	5.87	12.89	9.95	18.50	7.45	14.98
15 - 20 dB	5.74	12.98	9.83	**18.52**	7.41	15.06
20 - 20 dB	5.66	**12.99**	9.76	18.50	7.32	15.06

Table 6.25

Adapt SCE ext	m+m		m+f		all	
kmeans β - silence	SDR	SIR	SDR	SIR	SDR	SIR
hard - 0.0	4.51	11.05	7.23	13.19	6.35	13.24
hard - 20 dB	4.42	11.00	7.13	13.09	6.16	13.09
5 - 20 dB	4.71	9.54	8.14	13.92	5.33	8.89
10 - 20 dB	**4.71**	**10.89**	**8.28**	14.56	6.61	12.57
15 - 20 dB	4.66	11.08	8.27	14.62	**6.62**	13.16
20 - 20 dB	4.64	11.08	8.26	**14.64**	6.59	**13.31**

Table 6.26: Evaluation of the SCE method using the Adaptive layer finetuned with a chunks of size 30720

SCE rand 400	all	
kmeans β - silence	SDR	SIR
hard	7.42	15.53
hard - 20 dB	7.42	15.53
5 - 20 dB	7.67	12.73
10 - 20 dB	**8.01**	15.34
15 - 20 dB	7.84	15.59
20 - 20 dB	7.73	**15.65**

(a) Using spectrograms

Adapt SCE rand ext	all	
kmeans β - silence	SDR	SIR
hard	6.50	13.61
hard - 20 dB	5.91	13.46
5 - 20 dB	6.05	10.27
10 - 20 dB	**6.36**	13.22
15 - 20 dB	6.21	13.60
20 - 20 dB	6.14	**13.70**

(b) Using the Adaptive layer

Table 6.27: •

SCE+DANet 400	m+m		m+f		all	
kmeans β - silence	SDR	SIR	SDR	SIR	SDR	SIR
hard - 0.0	6.07	13.89	9.67	18.14	7.59	15.71
hard - 20 dB	5.51	13.64	8.94	17.59	6.92	15.27
5 - 20 dB	4.69	6.78	10.08	15.91	6.69	9.69
10 - 20 dB	6.75	11.69	**10.36**	18.26	8.41	14.21
15 - 20 dB	7.04	12.89	10.19	**18.33**	8.56	15.05
20 - 20 dB	**7.12**	**13.42**	10.08	18.30	**8.58**	**15.38**
25 - 20 dB	6.98	13.42	9.95	18.20	8.29	15.09

Table 6.28

Adapt SCE+DANet - ext	m+m		m+f		all	
kmeans β - silence	SDR	SIR	SDR	SIR	SDR	SIR
hard - 0.0	6.42	13.75	9.69	18.08	7.53	14.15
hard - 20 dB	6.30	13.69	9.44	17.82	7.35	14.99
5 - 20 dB	6.23	11.24	9.73	17.16	6.33	10.42
10 - 20 dB	**6.63**	13.53	**9.84**	18.11	7.79	14.61
15 - 20 dB	6.60	13.79	9.83	18.18	**7.79**	15.16
20 - 20 dB	6.58	**13.85**	9.82	**18.20**	7.76	**15.28**

Table 6.29: •

SCE ext + enh	m+m		m+f		all	
optimizer	SDR	SIR	SDR	SIR	SDR	SIR
SCE	9.27	14.96	12.04	18.88	9.94	15.53
SCE + NS	x	x	x	x	10.49	16.55
SCE + DANet	9.23	15.07	11.95	18.79	10.22	15.81
Adapt SCE	6.99	12.01	10.01	15.68	8.63	13.88
Adapt SCE + NS	x	x	x	x	8.60	13.79
Adapt SCE + DANet	8.36	13.87	11.06	17.58	9.26	14.76

Table 6.30

SCE finetuned	m+m		m+f		all	
method	SDR	SIR	SDR	SIR	SDR	SIR
SCE	-	-			11.30	17.34
SCE + NS	x	x	x	x	**11.59**	**18.02**
SCE + DANet	10.28	16.51	12.92	20.05	11.42	17.78
Adapt SCE						
Adapt SCE + NS	x	x	x	x		
Adapt SCE + DANet						

Table 6.31

Improvements in dB	m+m		m+f		all	
	SDR	SIR	SDR	SIR	SDR	SIR
DPCL [19]	4.0	-	9.07	-	6.54	-
DPCL (ours)	-	-	8.00	14.91	5.49	9.85
Adapt DPCL (ours)	-	-	**10.26**	**17.05**	**7.84**	**13.64**
DPCL - ext (ours)	-	-	10.43	17.97	4.74	9.28
Adapt DPCL - ext (ours)	-	-	10.97	19.56	8.50	15.79
DPCL - enh (ours)	-	-	10.14	16.09	7.59	12.34
Adapt DPCL - enh (ours)	-	-	10.64	15.81	8.76	12.81
DPCL++ [24]	9.4	-	12.0	-	10.08	-
DPCL++ - ext + enh (ours)	-	-	**12.38**	**19.18**	9.08	14.38
Adapt DPCL++ - ext + enh (ours)	-	-	11.34	17.78	**9.28**	**14.64**
DPCL++ - finetuned (ours)	-	-	-	-	-	-
Adapt DPCL++ - finetuned (ours)	-	-	-	-	-	-

Table 6.32: •

Improvements in dB	m+m / f+f		m+f		all	
	SDR	SIR	SDR	SIR	SDR	SIR
SCE [45]	5.48	-	9.98	-	7.69	-
SCE (ours)	-	-	7.69	14.47	5.89	12.23
SCE NS (ours)	x	x	x	x	6.67	12.80
Adapt SCE NS (ours)	x	x	x	x	6.25	11.27
SCE + DANet (ours)	-	-	-	-	-	-
Adapt SCE + DANet (ours)	5.78	11.51	9.05	15.67	7.37	13.27
SCE - ext (ours)	-	-	9.98	16.90	7.54	13.17
Adapt SCE - ext (ours)	-	-	8.28	14.56	-	-
SCE NS - ext (ours)	-	-	-	-	-	-
Adapt SCE NS - ext (ours)	-	-	-	-	-	-
SCE + DANet - ext (ours)	-	-	-	-	-	-
Adapt SCE + DANet - ext (ours)	-	-	-	-	-	-
SCE - ext + enh (ours)	-	-	12.04	18.88	-	-
Adapt SCE - ext + enh (ours)	-	-	-	-	-	-
SCE NS - ext + enh (ours)	-	-	-	-	-	-
Adapt SCE NS - ext + enh (ours)	-	-	-	-	-	-
SCE + DANet - ext + enh (ours)	-	-	-	-	-	-
Adapt SCE + DANet - ext + enh (ours)	-	-	-	-	-	-
SCE - finetuned (ours)	-	-	-	-	-	-
Adapt SCE - finetuned (ours)	-	-	-	-	-	-
SCE NS - finetuned (ours)	-	-	-	-	-	-
Adapt SCE NS - finetuned (ours)	-	-	-	-	-	-
SCE + DANet - finetuned (ours)	-	-	-	-	-	-
Adapt SCE + DANet - finetuned (ours)	-	-	-	-	-	-

Table 6.33: •

Chapter 7

Further Work and Discussion

We have seen that using the proposed Adaptive layer instead of spectrograms can lead to better results in some cases but still struggles to separate more than a 2 speakers mixture. Trying many other architecture possibilities for the Adaptive layer is one way to go to improve the reconstruction error. This reconstruction error is mostly due to the maxpooling layer that makes the front-end losing some information when computing the latent representation of the mixture. Another interesting path would be to construct such a network for multiple channel in input. An approach that could lead to better performances using spectrograms in input would be to use complex masks like described in [49] and used in [11]. Indeed, with such an approach the phase of the complex spectrogram is as well kept and since the whole information is fed to the deep neural architecture, this might lead to a better interpretation to produce more accurate embeddings and thus better separation.

Dilated Convolutional Neural Networks have as well shown to have very good results on Image Processing, and therefore using them before the BLSTM layers, like in [11], could extract some interesting features that the BLSTM are missing when outputting the (t,f) embeddings.

Since many other works on the multi speaker separation problem are using the World Street Journal (WSJ) dataset in order to train and evaluate there models, extending our work to be able to process this dataset would be good to compare it to many other methods.

One drawback of our approach is due to all the training phases necessary to converge to decent results in terms of SDR and SIR improvements. Another approach with the Adaptive layer would be not to apply any pretraining process but train it in end-to-end way directly with the source separation network. For instance, DANet [7] proposes a architecture capable to directly infer separated sources and therefore can be trained directly with Adaptive layer as a whole.

Chapter 8

Conclusion

To conclude, we have shown in this study that using an architecture such as the presented Adaptive layer can enhance the results of current state of the art methods for the multi speaker separation problem involving 2 speakers. Even though this layer is still not extendable to more than 2 speakers, these results are promising and further research in this direction could lead to a new network capable of fully reconstructing separated signals for many speakers. We have shown as well that the Source Contrastive Estimation can significantly be improved using methods such as combining it with the DANet objective function, using negative sampling and using techniques brought by the Deep Clustering method. We guess that a good extension of this work would be not to pretrain the Adaptive Layer like we did in this study but use it directly in an end-to-end way. A good approach would be to apply it to ADANet [**?**] network and train the whole network in one global step, this might lead to a better latent representation for the multi speaker separation problem.

References

[1] Martin Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, Manjunath Kudlur, Josh Levenberg, Rajat Monga, Sherry Moore, Derek G. Murray, Benoit Steiner, Paul Tucker, Vijay Vasudevan, Pete Warden, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. Tensorflow: A system for large-scale machine learning. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, pages 265–283, 2016.

[2] Triantafyllos Afouras, Joon Son Chung, and Andrew Zisserman. The conversation: Deep audio-visual speech enhancement. *CoRR*, abs/1804.04121, 2018.

[3] J. Allen. Short term spectral analysis, synthesis, and modification by discrete fourier transform. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 25(3):235–238, Jun 1977.

[4] Erich Paul Andrag. An autoencoder as a naive approach to audio source separation. 2015.

[5] Pierre Baldi. Autoencoders, unsupervised learning, and deep architectures. In Isabelle Guyon, Gideon Dror, Vincent Lemaire, Graham Taylor, and Daniel Silver, editors, *Proceedings of ICML Workshop on Unsupervised and Transfer Learning*, volume 27 of *Proceedings of Machine Learning Research*, pages 37–49, Bellevue, Washington, USA, 02 Jul 2012. PMLR.

[6] Albert S. Bregman and Stephen McAdams. Auditory scene analysis: The perceptual organization of sound. *The Journal of the Acoustical Society of America*, 95(2):1177–1178, 1994.

[7] Zhuo Chen, Yi Luo, and Nima Mesgarani. Deep attractor network for single-microphone speaker separation. *CoRR*, abs/1611.08930, 2016.

[8] Martin Cooke. *Modelling Auditory Processing and Organisation*. Cambridge University Press, New York, NY, USA, 1993.

[9] Anthony D'Amato. Adaptive-multispeaker-separation. `https://github.com/Totoketchup/Adaptive-MultiSpeaker-Separation`, 2018.

[10] Daniel P. W. Ellis. *Prediction-driven Computational Auditory Scene Analysis*. PhD thesis, Cambridge, MA, USA, 1996. AAI0597425.

[11] Ariel Ephrat, Inbar Mosseri, Oran Lang, Tali Dekel, Kevin Wilson, Avinatan Hassidim, William T. Freeman, and Michael Rubinstein. Looking to listen at the cocktail party: A speaker-independent audio-visual model for speech separation. *CoRR*, abs/1804.03619, 2018.

[12] Aviv Gabbay, Ariel Ephrat, Tavi Halperin, and Shmuel Peleg. Seeing through noise: Speaker separation and enhancement using visually-derived speech. *arXiv preprint arXiv:1708.06767*, 2017.

[13] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[14] Emad M. Grais and Mark D. Plumbley. Single channel audio source separation using convolutional denoising autoencoders. *CoRR*, abs/1703.08019, 2017.

[15] Emad M. Grais, Gerard Roma, Andrew J. R. Simpson, and Mark D. Plumbley. Discriminative enhancement for single channel audio source separation using deep neural networks. *CoRR*, abs/1609.01678, 2016.

[16] Emad M. Grais, Mehmet Umut Sen, and Hakan Erdogan. Deep neural networks for single channel source separation. *CoRR*, abs/1311.2746, 2013.

[17] Jiuxiang Gu, Zhenhua Wang, Jason Kuen, Lianyang Ma, Amir Shahroudy, Bing Shuai, Ting Liu, Xingxing Wang, and Gang Wang. Recent advances in convolutional neural networks. *CoRR*, abs/1512.07108, 2015.

[18] Y. Han, J. Kim, and K. Lee. Deep convolutional neural networks for predominant instrument recognition in polyphonic music. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 25(1):208–221, Jan 2017.

[19] John R. Hershey, Zhuo Chen, Jonathan Le Roux, and Shinji Watanabe. Deep clustering: Discriminative embeddings for segmentation and separation. *CoRR*, abs/1508.04306, 2015.

[20] Geoffrey Hinton, Nitish Srivastava, and Kevin Swersky. Neural networks for machine learning lecture 6a overview of mini-batch gradient descent.

[21] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Comput.*, 9(8):1735–1780, November 1997.

[22] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359 – 366, 1989.

[23] Po-Sen Huang, Minje Kim, Mark Hasegawa-Johnson, and Paris Smaragdis. Deep learning for monaural speech separation. pages 1562–1566, 05 2014.

[24] Yusuf Isik, Jonathan Le Roux, Zhuo Chen, Shinji Watanabe, and John R. Hershey. Single-channel multi-speaker separation using deep clustering. *CoRR*, abs/1607.02173, 2016.

[25] Jaeyoung Kim, Mostafa El-Khamy, and Jungwon Lee. Residual LSTM: design of a deep recurrent architecture for distant speech recognition. *CoRR*, abs/1701.03360, 2017.

[26] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014.

[27] Morten Kolbæk, Dong Yu, Zheng-Hua Tan, and Jesper Jensen. Multi-talker speech separation and tracing with permutation invariant training of deep recurrent neural networks. *CoRR*, abs/1703.06284, 2017.

[28] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521:436 EP –, May 2015.

[29] Daniel D. Lee and H. Sebastian Seung. Algorithms for non-negative matrix factorization. In *Proceedings of the 13th International Conference on Neural Information Processing Systems*, NIPS'00, pages 535–541, Cambridge, MA, USA, 2000. MIT Press.

[30] Xugang Lu, Yu Tsao, Shigeki Matsuda, and Chiori Hori. Speech enhancement based on deep denoising autoencoder. In *INTERSPEECH*, pages 436 – 440, 2013.

[31] Yi Luo, Zhuo Chen, John Hershey, Jonathan Le Roux, and Nima Mesgarani. Deep clustering and conventional networks for music separation: Stronger together. 2017:61–65, 03 2017.

[32] Yi Luo and Nima Mesgarani. Tasnet: time-domain audio separation network for real-time, single-channel speech separation. *CoRR*, abs/1711.00541, 2017.

[33] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean. Distributed representations of words and phrases and their compositionality. *CoRR*, abs/1310.4546, 2013.

[34] Stylianos Ioannis Mimilakis, Konstantinos Drossos, Tuomas Virtanen, and Gerald Schuller. A recurrent encoder-decoder approach with skip-filtering connections for monaural singing voice separation. *CoRR*, abs/1709.00611, 2017.

[35] Ryosuke Okuta, Yuya Unno, Daisuke Nishino, Shohei Hido, and Crissman Loomis. Cupy: A numpy-compatible library for nvidia gpu calculations. In *Proceedings of Workshop on Machine Learning Systems (LearningSys) in The Thirty-first Annual Conference on Neural Information Processing Systems (NIPS)*, 2017.

[36] Vassil Panayotov, Guoguo Chen, Daniel Povey, and Sanjeev Khudanpur. Librispeech: An asr corpus based on public domain audio books. *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 5206–5210, 2015.

[37] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the difficulty of training recurrent neural networks. In *Proceedings of the 30th International Conference on International Conference on Machine Learning - Volume 28*, ICML'13, pages III–1310–III–1318. JMLR.org, 2013.

[38] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Neurocomputing: Foundations of research. chapter Learning Representations by Back-propagating Errors, pages 696–699. MIT Press, Cambridge, MA, USA, 1988.

[39] Mikkel N. Schmidt and Rasmus Kongsgaard Olsson. Single-channel speech separation using sparse non-negative matrix factorization. In *INTERSPEECH*, 2006.

[40] Jonathan Shen, Ruoming Pang, Ron J. Weiss, Mike Schuster, Navdeep Jaitly, Zongheng Yang, Zhifeng Chen, Yu Zhang, Yuxuan Wang, R. J. Skerry-Ryan, Rif A. Saurous, Yannis Agiomyrgiannakis, and Yonghui Wu.

Natural TTS synthesis by conditioning wavenet on mel spectrogram predictions. *CoRR*, abs/1712.05884, 2017.

[41] Andrew J. R. Simpson. Probabilistic binary-mask cocktail-party source separation in a convolutional deep neural network. *CoRR*, abs/1503.06962, 2015.

[42] Andrew J. R. Simpson. Time-frequency trade-offs for audio source separation with binary masks. *CoRR*, abs/1504.07372, 2015.

[43] Andrew J. R. Simpson, Gerard Roma, and Mark D. Plumbley. Deep karaoke: Extracting vocals from musical mixtures using a convolutional deep neural network. *CoRR*, abs/1504.04658, 2015.

[44] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15:1929–1958, 2014.

[45] Cory Stephenson, Patrick Callier, Abhinav Ganesh, and Karl S. Ni. Monaural audio speaker separation with source contrastive estimation. *CoRR*, abs/1705.04662, 2017.

[46] Aäron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew W. Senior, and Koray Kavukcuoglu. Wavenet: A generative model for raw audio. *CoRR*, abs/1609.03499, 2016.

[47] Shrikant Venkataramani and Paris Smaragdis. End-to-end source separation with adaptive front-ends. *CoRR*, abs/1705.02514, 2017.

[48] E. Vincent, R. Gribonval, and C. Fevotte. Performance measurement in blind audio source separation. *IEEE Transactions on Audio, Speech, and Language Processing*, 14(4):1462–1469, July 2006.

[49] DeLiang Wang and Jitong Chen. Supervised speech separation based on deep learning: An overview. *CoRR*, abs/1708.07524, 2017.

[50] Yuxuan Wang, Arun Narayanan, and DeLiang Wang. On training targets for supervised speech separation. *IEEE/ACM Trans. Audio, Speech and Lang. Proc.*, 22(12):1849–1858, December 2014.

[51] Z. Q. Wang and D. Wang. Recurrent deep stacking networks for supervised speech separation. In *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 71–75, March 2017.

[52] Zhong-Qiu Wang, Jonathan Le Roux, and John R. Hershey. Alternative objective functions for deep clustering.

[53] M. Weintraub. The grasp sound separation system. In *ICASSP '84. IEEE International Conference on Acoustics, Speech, and Signal Processing*, volume 9, pages 69–72, Mar 1984.

[54] Mitchel Weintraub. *A Theory and Computational Model of Auditory Monaural Sound Separation (Stream, Speech Enhancement, Selective Attention, Pitch Perception, Noise Cancellation)*. PhD thesis, Stanford, CA, USA, 1985. AAI8602565.

[55] Felix Weninger, Hakan Erdogan, Shinji Watanabe, Emmanuel Vincent, Jonathan Roux, John R. Hershey, and Björn Schuller. Speech enhancement with lstm recurrent neural networks and its application to noise-robust asr. In *Proceedings of the 12th International Conference on Latent Variable Analysis and Signal Separation - Volume 9237*, LVA/ICA 2015, pages 91–99, Berlin, Heidelberg, 2015. Springer-Verlag.

[56] Y. Xu, J. Du, L. R. Dai, and C. H. Lee. An experimental study on speech enhancement based on deep neural networks. *IEEE Signal Processing Letters*, 21(1):65–68, Jan 2014.

[57] Dong Yu, Xuankai Chang, and Yanmin Qian. Recognizing multi-talker speech with permutation invariant training. *CoRR*, abs/1704.01985, 2017.

[58] Dong Yu, Morten Kolbæk, Zheng-Hua Tan, and Jesper Jensen. Permutation invariant training of deep models for speaker-independent multi-talker speech separation. *CoRR*, abs/1607.00325, 2016.

[59] Matthew D. Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. *CoRR*, abs/1311.2901, 2013.